Project report for the course Multisensory Interactive Systems Academic Year: 2023-2024

Fishing Rod

Marco Bortolotti, marco.bortolotti-1@studenti.unitn.it, 247431

Riccardo Calcagno, riccardo.calcagno@studenti.unitn.it, 247012

Jan Kompatscher, jan.kompatscher@studenti.unitn.it, 247017

Manuel Righele, manuel.righele@studenti.unitn.it, 239945

Abstract

This research study examines the variation in perception and measured performance when engaging in a multifaceted and integrative activity, such as fishing, depending on the controlled variable: specific subset of sensory modalities activated. By inspecting the differences in the outcomes of partial and total multisensory integration, we aim to demonstrate the presence of the Superadditive effect, even in the higher levels of abstraction of Human-Computer interaction.

In particular, we decided to develop a fishing game that teaches the users how to catch the fish correctly, thanks to specific feedback. More precisely, the system takes as input the data coming from sensors fixed on the rod prototype. They are sent to the computer via WiFi and then processed to extract the information related to the movement. Based on this knowledge, the algorithm returns real-time feedback to the user. This stimulus can be auditory, haptic, visual, or a combination of them, depending on the current kind of investigation. As fishing is a complex task that requires a good level of concentration and experience, performance could be different depending on the kind of feedback retrieved from the environment and the number of accumulated attempts. All these aspects will be discussed in detail in the following chapters.

Project Repository: https://github.com/RiccardoCalcagno/MultisensoryFishingRod

Table of Contents

1. Introduction

The study related to multisensory perception aims to understand how the brain integrates and elaborates information coming from more than one sensory channel to produce a coherent experience. Indeed, modern science has revealed that the senses are connected and interact with each other. Thus, stimuli that affect different human sensory systems cause different perceptions of a particular event or object in respect to an unimodal stimulus.

For example, if we think about talking in a noisy environment, the single auditory signal could not be sufficient to understand the speaker. It's more likely to get the message if the listener also exploits the visual information from the lips of the speaker. Thus, it is noticeable that the combination of two or more different stimuli generates a stronger signal rather than the single one.

For these reasons, designers of interactive systems can exploit the characteristics of the human being to obtain a particular behavior of the user in response to provided stimuli. For instance, the arrival of a call on the cell phone is characterized by the ringtone, the feeling of a vibration and the appearance of a pop-up on the screen. In this case, the objective of the designer is to capture our attention and therefore make us have a look at the phone.

Following this reasoning, in our research we wanted to investigate how to make the user solve a specific task by providing them a stream of stimuli from different sensory channels. More precisely, we decided to develop a system to simulate the fishing experience in the most realistic way possible. This choice is based on the fact that fishing is a complex task which requires a high level of attention and a good level of experience. Thus, the user has to be really careful of stimuli coming from the system in order to achieve the final goal of catching the fish.

Furthermore, we are confident that the performance of users can improve over time as far as the number of attempts they reach.

In our scenario, the individual is provided with a defined set of stimuli chosen a-priori based on the kind of investigation. In particular, the system can perform visual, auditory, and haptic feedback and these perceptions have been analyzed in groups of 2 or all together:

- auditory plus haptic
- auditory plus visual
- haptic plus visual
- visual plus haptic plus auditory

In the context of fishing, we are confident that the summation of all these three feedbacks is the most effective combination to catch the fish. However, we wanted to conduct a comprehensive analysis to understand what are the most salient feedbacks to increase the probability of success, reducing the number of mistakes and the time. In particular, we suppose that the visual stimulus has a strong relevance in understanding if the fish is interested in eating the bait while the auditory and the haptic feedback help the user in understanding if the fish is hooked, reducing the reaction time.

To make the experience even more realistic and immersive, we developed a face recognition system which moves the camera inside the GUI following the head movements of the user. In addition, the algorithm generates a real-time sound of the rod components such as the rotation of the fishing reel and the movements of the rod.

Then, another aspect we considered in the design of our experiment was to arouse emotions in the user. In these terms, the system provides different kinds of music based on the status of the game. For instance, if the fish is hooked, the system will perform a music related to happiness while if the fish is lost, the sound will reflect defeat. We supposed that this aspect can help the probability of success of users.

Is noticeable that the experiment is not trivial since it takes into account several aspects related to interaction between humans and machines. The system works thanks to the wireless communication between a Processing instance running on a computer and an ESP32 fixed on the rod which is connected to sensors and actuators. In particular, we installed a set of ERMs on the rod bottom where users should hold the rod, to provide different haptic sensations as the fish touches or hooks the bait. Then, we developed a Processing GUI in which users can see the fish under the water exploiting the face tracking, and integrate the sound information thanks to Pure Data which generates real-time sound.

In the next chapters we will explain in detail the design of our product starting from the assumptions we made and then describing all the components which interact to make the system work properly. Then we have reported all the experiments we did to understand which kinds of stimuli improve the performance of users in catching the fish.

2. Related Work

The human-machines interaction is the study of the interaction between human beings and computers which aims to design and develop interactive systems for supporting users in their activities. This concept is strictly correlated to how the human brain elaborates the information from different sensory channels to return the perception of a particular event or object. Exploiting this peculiarity, researchers proposed several approaches which combine different kinds of actuators to cause sensations and illusion or to lead users to perform a specific action. For instance, generating a rough sound while the user is rubbing his or her hands results in the so-called Parchment Skin Illusion [14]. Another example is the combination between the auditory and the haptic feedback to give blind people information about the surrounding environment.

In general, interactive systems have a very wide range of applications such as robotics, medical, virtual reality, music, communication, and gaming.
Related to the latter, we found out that Nintendo developed a game called Wii Play [13] in which one of the possible mini-games was fishing. In this case, the Wii exploited the infrared signal coming from the remote control to understand users' movements.

Furthermore, it also gave haptic feedback based on the fish actions and a graphical animation of the fishes inside the pond. In this case, the only variable that is taken into account by the game is the time. Indeed, the probability of success depends only on the timing of the user in making a strong shake when the fish is hooked.

Nevertheless, our objective focuses on developing a game which should be as realistic as possible.

Firstly, we built a real prototype of the rod which also includes the fishing reel. Thus, the user can deal with a tangible object which has the same characteristics as a real rod.

Secondly, the system takes into account different aspects of the fishing such as the movements to attract the fish, the power and the timing of the shake when the fish is hooked, and the speed of the wire when it starts to be retrieved. Thus, it becomes not so trivial catching the fish. However, we are confident that the performance of the user could be improved over time.

Finally, we also wanted to develop a system that is immersive and arouses emotions. For these reasons, we developed a face recognition system that moves the camera inside the GUI as far as the head movements. Then, the system generates a specific kind of music based on the status of the game.

3. Architecture design

3.1. Assumptions

- We assume that, to enjoy the game to its full potential, the **environment** must be high and **large** enough to allow the user to bring and move the rod without **any obstacles**.
- In the experiment **sound** is generated to give feedback to users action and **to arouse emotions**. Thus, it's preferable to play in a non-noisy environment.
- It's important to be sure that the connection is secure since anyone connected to the network can send UDP packets to the computer.
- We want to make this experience more fascinating and immersive and therefore it's fundamental to have a wide screen on which to project the visual interface.
- Since **fishing** is a very **common activity,** we suppose that testing our product could be a positive experience for both people who want to approach **fishing for the first time** and people who just want to **have fun** playing this game.

3.2. Recommendations and Suggestions

- It's important to mention that in the game users have to bring the rod which is quite heavy and long and therefore it is **not recommended for** those people who have any physical impairment.
- We strongly suggest a projector instead of the screen of the laptop to also enjoy the feature of the face tracker.

3.3. Components of the Architecture

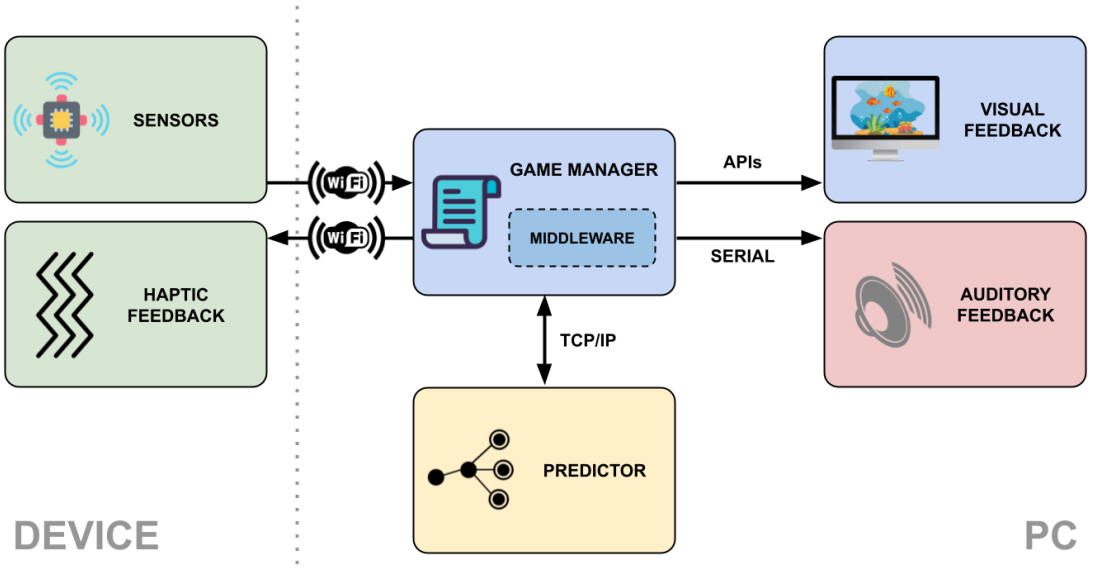
First and more salient component of our architecture is our tangible object: the fishing rod.

It communicates with a PC which runs a middleware, a predictor, a game manager and the visual and auditory modules.

Every component acts for different needs:

- **Game manager:** is the core part of the project: it can make decisions regarding the behavior of the program, the rules and the scores of the game. Every data flow of the software passes through it. It is done with the Processing framework and is composed of different modules.
- **Middleware:** allows the **communication** of the various components of the software and hardware programmed in heterogeneous programming languages, through different types of channels: TCP/IP stack, serial ports and internal APIs of the software.
- **Physical device:** tracks the movements and the control of the fishing rod, and gives feedback vibrations in the handle;
- **Visual module:** takes instruction from the game manager and provides a video stream of the 3D scenes where there is the fish. A webcam is used to track the position of the eyes of the user to make the game more immersive, with OpenFace2.0 [15.].
- **Auditory module:** provides different types of sounds according to the actions that the user makes with the fishing rods; it also controls the music of the game, which can be more emotionally arousal/intensive in some situations, and more relaxed/distensive in others. It is developed with Pure Data.
- **Predictor:** is another part of the software that implements some functions to recognize patterns in accelerometer data that are the main and essential movements of the fishing. It uses some data analysis methods to filter, and then a Tensorflow model to classify the movements. It is done in Python 3.

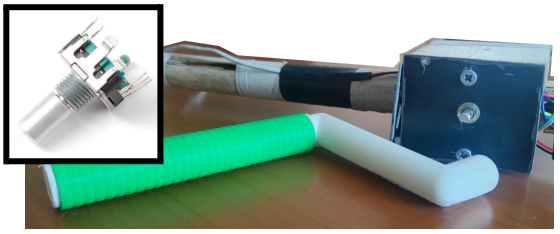
Here is a **diagram** and a detailed description of how the system works, connects various devices and coordinates tasks to have a global synchronized application.



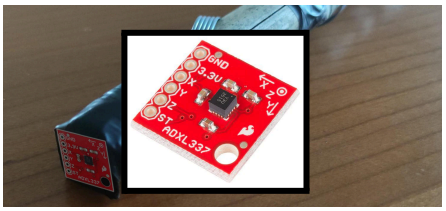
In this figure, we can see that five components run in the PC, while only one component in the external device, the fishing rod. Every color of the diagram represents a different program in a different language. Another notation before going into the details of the components: the connections permit communication that is virtual inside the PC (TCP/IP or Serial) and physical (WiFi 2.4 GHz) with the device, through a UDP channel; finally they can be unidirectional or bidirectional.

The fishing rod is developed with *Arduino IDE* and contains a **board ESP32-CAM** that controls all the sensors and actuators (haptic feedback):

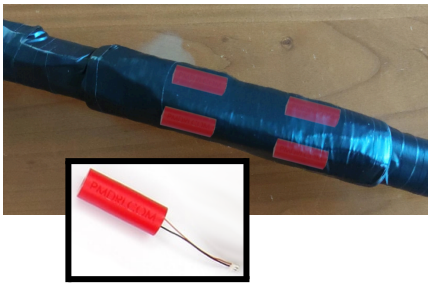
- **Rotary encoder**, to get the velocity and direction of the reel of the fishing rod;



- **Accelerometer** on the top of the fishing rod, to get its movements, (the gyroscope was not necessary for our purposes);

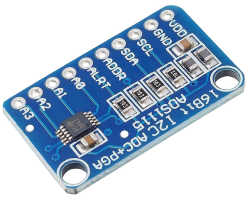


- Four **vibration motors** on the handle, to get feedback from the game manager.

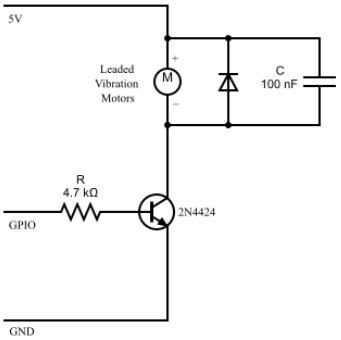


There are also **two other support boards** to enable the proper operation:

- **ADS1115 converter**, it reads up to four ports with ADC and communicates with the main board through Soft I2C, because the ADC1 of the main board is not reachable, and ADC2 is used by WiFi module; it is used to read X, Y, Z of the accelerometer;



- Self-developed **ERM's driver**, with EMI suppression, to permits conversion from the GPIO at 3.3V to a 5V power pins, with a MOSFET transistor, a diode, a capacitor, and a resistor, all tinned in a "millefori".



3.4 Usage model

The user will hold the handle of the fishing rod (as with a real one) with the wheel facing to the right. Placed as closed as possible to the camera, without the risk to hit it the experience ca begin.

After the start of the processing app, it will be asked to the user to digit his/her name and select which modalities to activate during the experience: haptic, video and/or audio.

Once the experience is started, a non-playable character (NPC), the fish, is spawned in the space. The same space where virtually the rod and its wire are interacting.

At first, the fish is not very interested in the bait and swims randomly around the screen. The user has to perform long attracting "shakes" which are figure-8 movements or little attracting shakes to lure in the fish. Movements that are too large and intense scare the fish away.
With increased interest the fish will do more and more directed approaches. This will be visible on the screen and audible in change of the music's number of instruments and volume.

When it has large enough interest, the fish will take bites at the bait. The bites will be visible, audible, and haptically feelable. The user has to perform a strong hooking shake in the correct way (straight up, wide, short-in-time and intense) at the correct timing. If it's the wrong movement or bad timing the fish is not hooked and loses a bit of its accumulated interest.

In the contrary case, the fish will be hooked which again will be visible, audible, and haptically feelable. It can now be reeled in.

If it swims in the direction of the surface it can be reeled in with no problem, but if it swims to the bottom and the user keeps spinning the reel the wire tension is increased. Too high wire tension is again indicated with all three sensory stimuli (visual, auditory, haptic). At a certain threshold the fish is lost and the game is over.
On the other hand, if the fish is completely reeled in without ever overstepping the wire tension threshold the game is won.

4. Implementation

4.1 Tools

- **Pure Data** [18.]: open source visual programming language for multimedia used to write and execute our auditory module;
- **ArduinoIDE**: Used to write the code to upload in the board of our fishing rod;
- **Processing**: Used to write the code of our application;
- **Virtual Serial Ports** [17.]: Used to create the serial ports necessary for the CameraMovement module;
- **OpenFace2.0** [15.] software that already implemented a ML algorithm to do face recognition;
- **Anaconda, JupiterNotebook, VS Code**: Used to deploy and program in python;
- **GitHub**: For storage, Version control and presentation of our suite;
- **Draw.io**: to sketch our diagrams (also in UML 2.5);
- **Ultimaker**: to print our reel.

4.2 Libraries

— Processing —

- **java.io**: Used to write our measurements on FileSystem, Either for debug and evaluation.
- Open source utility to **boost PVector class** [16.]
- **processing.serial**, **processing.net**, **java.net**: Libraries that allow as to adopt Soket to perform TCP/IP communications in WLAN, DatagramSocket for UDP ones in WLAN and serial communication on virtual ports COM1/COM2 in local.
- **g4p_controls**: to create a rudimental GUI for standard WIMP applications in our app, (for the initial 2D setting of the game).

— Arduino —

- **WiFi.h**: library to connect to the WiFi, get IPs, and exchange data;
- **AsyncUDP.h**: library to setup a server and a client with UDP connections;
- **ADS1X15.h**: library to use the ADS1115 board (includes Wire.h that allows a Soft I2C on ports not designed for this purpose).

— PureData —

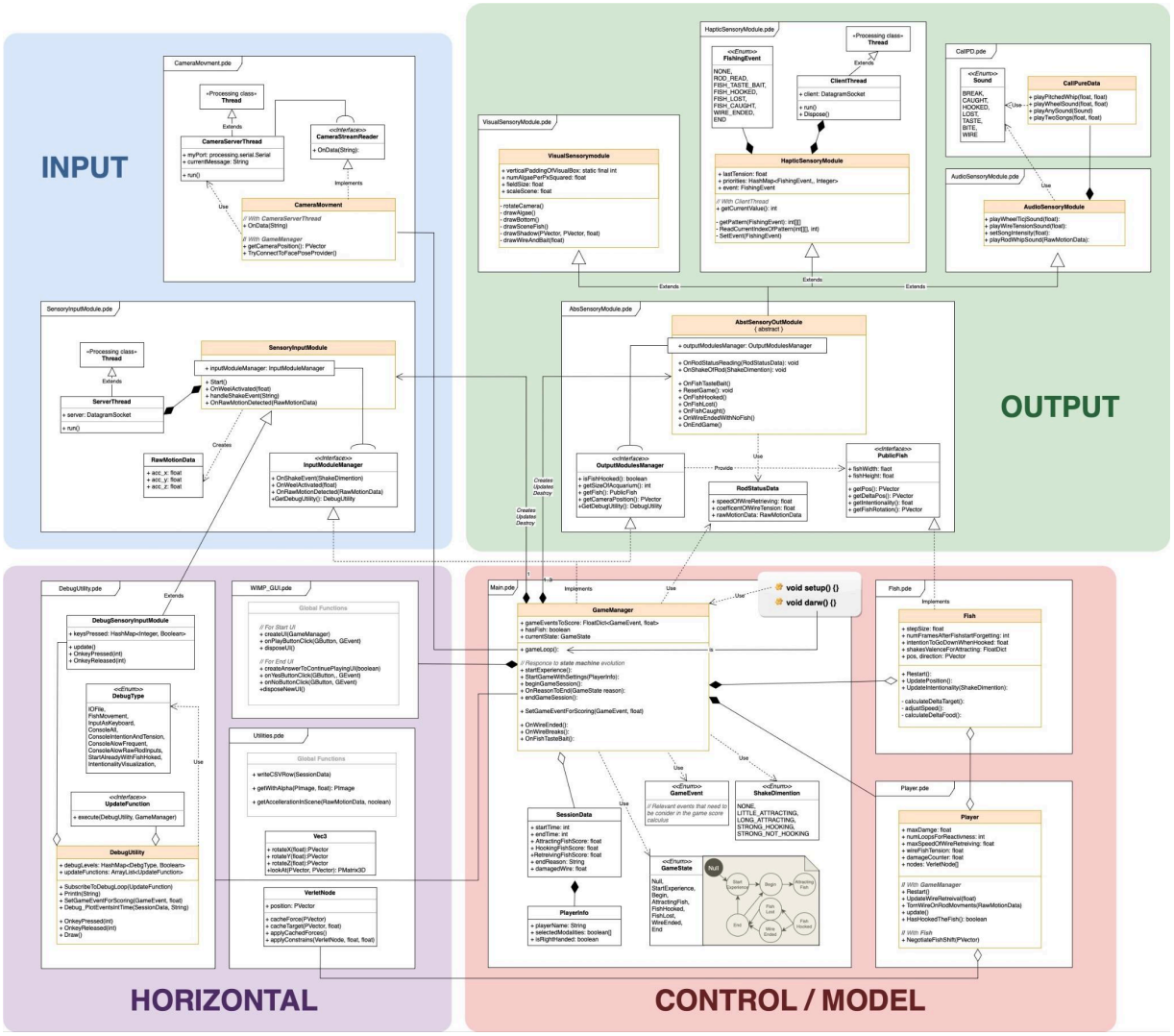
- **SDT library** : used to simulate sound of the rotation of the wheel, the sound of the water moving, and the tension of the wire.

— Python —

- **pyzmq**, **pyserial**, **keyboard**, **psutil**, **pandas**, **tensorflow, zmq, numpy, soket, os, serial, threading, subprocess.**

4.3 Processing Software

The code written in Processing language to coordinate all project functionalities is located in the directory: `MultisensoryFishingProcessing` within the project's root.



Class Diagram

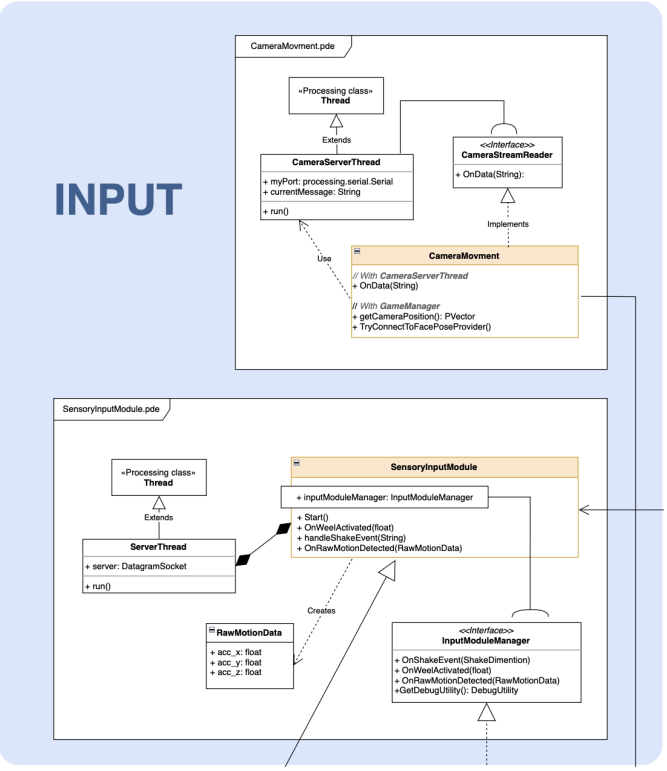
The class diagram above describes all the scripts in the directory, represented by white rectangles labeled with the file name.

A quick glance at the diagram will help understand the scope of responsibilities of this package. The following macro areas of interest are highlighted:

- **INPUT**: Scripts responsible for **collecting inputs** from **local** components (such as the facial recognition process) and **remote** components (from the fishing rod).
- **OUTPUT**: Scripts responsible for the multisensory representation of the fishing simulation, also providing direct and processed feedback from inputs.
- **CONTROL/MODEL**: Scripts responsible for managing the application's **lifecycle**, handling its **game loop** to **centralize** and process **inputs**, and extracting appropriate **game events** to assign them to output modules generically. They are also responsible for defining **model** classes for the proper and decoupled communication of classes.
- **HORIZONTAL**: Scripts responsible for horizontal functionalities **across the entire project**, such as utility classes, the *DebugUtility* class, and game score management.

Below, we will describe these areas of interest, delving into detail in some cases to describe some of the more interesting specifics.

4.3.1 Input



The dedicated script package for processing input signals includes:

1. `CameraMovement.pde`
2. `SensoryInputModule.pde`

The ***CameraMovement*** class is solely responsible for maintaining its cameraPosition field. It utilizes a dedicated thread to avoid burdening the main thread, retrieving the **user's face position in space** from an active subprocess of the open-source suite *OpenFace2.0* [15]. The face position, remapped in the game scene, is then retrieved by the main controller and passed to the visual output module to **move the camera in the 3D scene**, enhancing the perception of immersion in the simulation.

Similarly, the ***SensoryInputModule*** class employs a new thread for the same reason, but this time, it responds to inputs from the fishing rod.

> *More precisely, the data does not come directly from the ESP32-CAM board of the rod but from a local Python process acting as a middleman, listening to the board in the WLAN. This process adds shake predictions using tiny ML algorithms. Refer to the "Predictor" section for details.*

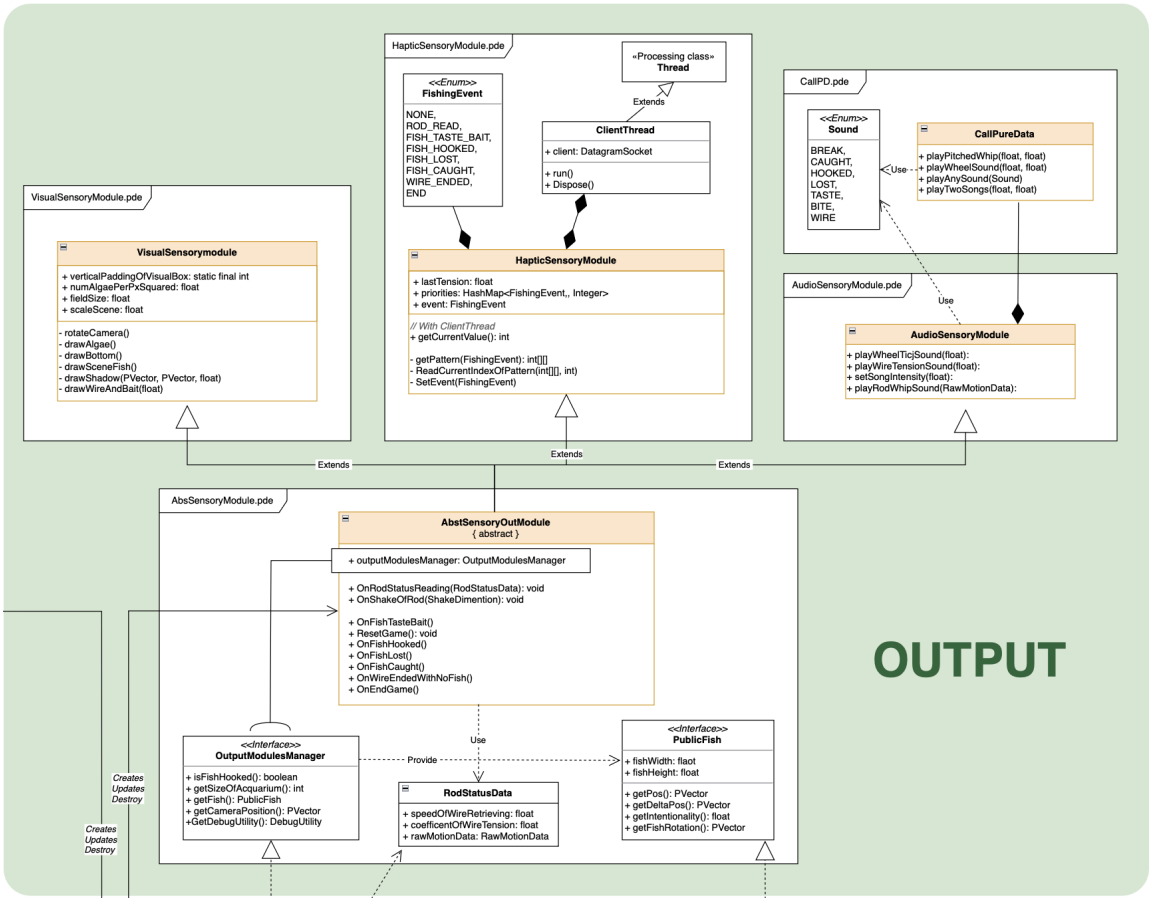
The input class requires an interface (implemented by the *GameManager*) through which it can transmit data from the rod in a decoupled and clean manner. The transmitted data includes:

- **Accelerations x, y, z:** Real numbers from the rod's accelerometer, aligned with the board's sampling frequency.
  Remapped as normalized accelerations projected onto the Cartesian axes of the game scene.
  Transmitted by invoking: *OnRawMotionDetected(RawMotionData)*

- **Rotary encoder rotation:** A representative number of the reel's speed and rotation, also aligned with the sampling frequency.
  Remapped as a normalized real in the range (-1, 1).
  Transmitted by invoking: *OnWeelActivated(float)*

- **Start/Continuation/End of a Shake type:** The predictor communicates the presence or absence of a shake type to the input module thread every 0.5 ms prediction window, except for STONG_HOOKING, which is instantaneous (thanks to a specific identification strategy).
  Remapped as values of an application domain enum *ShakeDimension*.
  Transmitted by invoking: *OnShakeEvent(ShakeDimension)*

4.3.2 Output



The dedicated output script package includes the following files:

1. `AbstSensoryModule.pde`
2. `VisualSensoryModule.pde`
3. `HapticSensoryModule.pde`
4. `AudioSensoryModule.pde`
5. `CallPD.pde`

From a Software Engineering perspective, the architecture of this class group demonstrates a significant effort in the initial stages of development to maintain the decoupling of control/model from the view (interpreted as the multisensorial representation of the output).

This decision was not only made to adhere to the MVP paradigm but, more importantly, to ensure the adherence to our foundational research assumption, deeply entrenched in the unalterable constraints of the hereditary principle:

> *We must strive, to the greatest extent possible, to provide a redundant representation of the output in different modalities when one is missing*

Having written the abstract class: *AbstSensoryModule*, a class that clearly indicates all game events that need to be listened to, forces our already difficult coordinated work to align with and comply with this assumption.

The *AbstSensoryModule* class provides each module with the public field *outputModuleManager* (a public interface of *GameManager*), which in turn allows access to the public interface of the fish *PublicFish* via *getFish()*. These two objects serve as the system model for the output modules, to be queried at will in every instance (**PULL**), retrieving information such as fish intent, position, and direction, whether the fish is hooked, camera position, and the size of the animated space. Conversely, to subscribe to game events triggered by the main controller (**PUSH**), modules must override specific methods. We distinguish three types of methods based on invocation frequency:

- **Asynchronous sporadic handlers**: Es. *OnFishTasteBait()*, *ResetGame()*, *OnFishHooked()*, *OnFishLost()*, … , *OnEndGame()*. These methods are invoked without regularity, occurring at the specific event they reference.
- **Game loop handler**: *OnRodStatusReading(RodStatusData)* is the method containing all information about the rod's state at the moment. It is invoked once per game loop cycle.
- **Start/End handler**: *OnShakeOfRod(ShakeDimention)* is the method invoked at the beginning of a shake type with an appropriate enum code and called again with a value of NONE at the end of the shake. (All shakes are concluded with a NONE, and redundant values are not dispatched).

Every effort has been made to make the experience as redundant as possible in each sensory mode, except for the representation of fish intent (not perceptible haptically to avoid overstimulating the channel). The following table describes how each event is connected to an appropriate output in each mode.

Event	Output
Fish Intentionality (stream)	● **Visual**: the fish get closer to the bait and the overall color of the video stream are getting brighter ● **Auditory**: The background music becomes louder and colored with more instrument tracks as the intentionality increases.
The fish taste the bait (single event)	● **Visual**: The fish open and close the mouth ● **Auditory**: Sound of a bite ● **Haptic**: little, sudden, impulse of vibration
The fish got hooked (single event)	● **Visual**: the fish remains attached to the wire ● **Auditory**: sound of a stronger bite ● **Haptic**: Turbulent and strong pattern of vibrations
Wire tension (stream)	● **Visual**: the wire color shift from black to red ● **Auditory**: sound of a rope that is stretching at the increase of the tension ● **Haptic**: a continuous vibration increase at the increase of wire tension
End of the game (single event)	● **Visual**, **Auditive**, **Haptic**: Expressive animations particular for the type of game ending

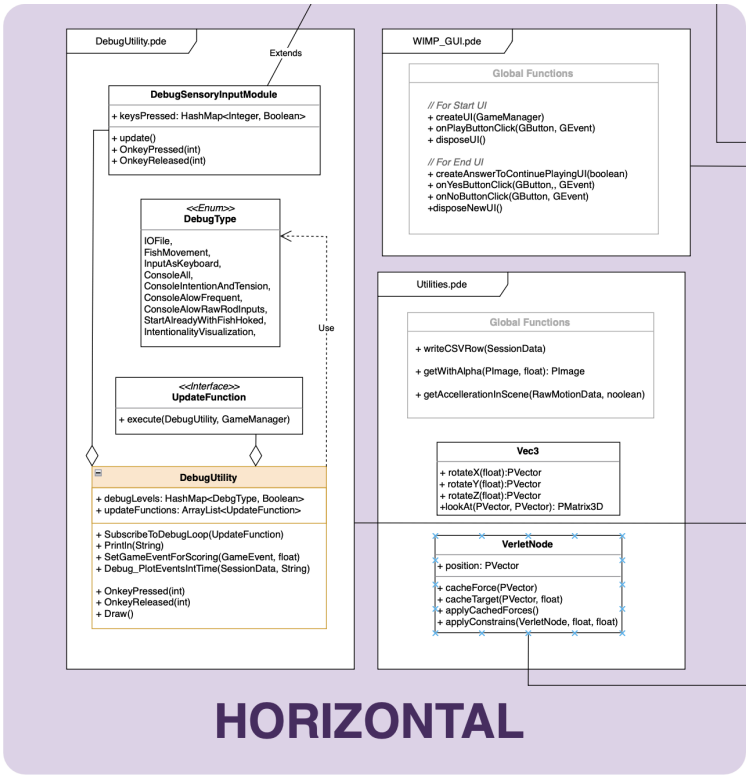
4.3.3 Horizontal

The ***DebugUtility*** class is extensively utilized to expedite debugging, testing, and fine-tuning processes, ensuring efficiency and clarity.

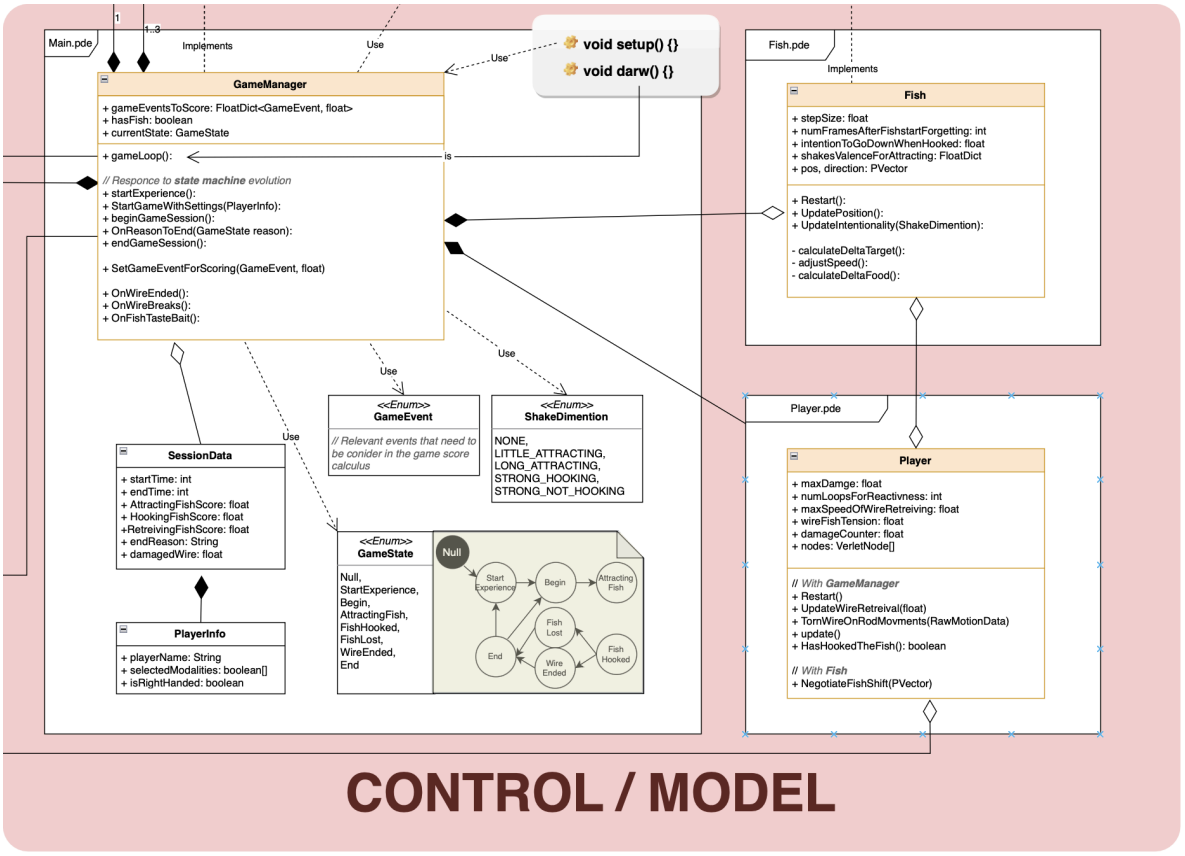
It incorporates a *HashMap* named *debugLevels* (refer to the code snippet in the appendix), allowing the programmer to enable or disable specific debug behaviors.

The ***DebugSensoryInputModule*** class (extending *SensoryInputModule*) facilitates the use of the keyboard to perform major input operations typically executed with the fishing rod in production.

The utility script, ***Utilities.pde***, encompasses global functions employed in creating a rope adhering to the laws of physics and spatial transformations in a 3D environment (see the appendix).



4.3.4 Control / Model



The package of scripts dedicated to control and modeling includes:

1. Main.pde
2. Fish.pde
3. Player.pde

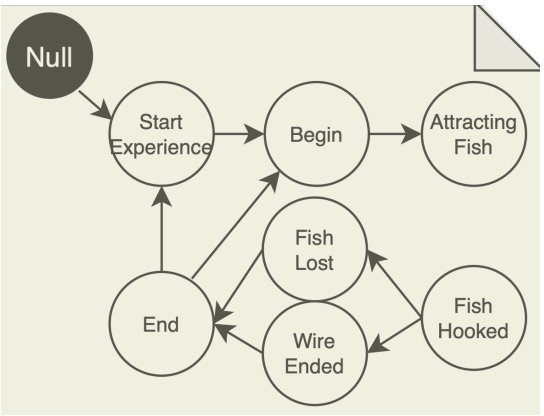
GameManager

The *GameManager* is responsible for the following duties:

1. Life Cycle control

Thanks to having designed a simplified version of a state machine with states defined by values of the enum: ***GameState***, the game controller manages centrally the transition from a state to another, also it notifies all the modules involved and responsive to those changes.

The state machine is illustrated by the following diagram:



2. Game Loop

The *GameManager* defines for all applications the game loop in which the data from the input modules are collected, in which those same data are re-elaborated in relation to entities like *Player* and *Fish*. And define the rhythm with which the output modules are rendering their streams.

3. Centralized handler of asynchronous events

The *GameManager* with its interface *InputModuleManager* is allowing entities like the *SensoryInputModule*, *Player* and *Fish*. To trigger some specific asynchronous events of the game logic. Like *OnFishTasteBait()*, *OnWireBreaks()*, *OnWireEnded()*, *OnShakeEvent(ShakeDimention)*, *OnWeelActivated(float)*, all of them have strategies to make their changes to be handled in the synchronized flow of the game loop.

4. Model guarantor

The GameManager is widely using structure of data that can be recognized by either the output and input modules, like ShakeDimention and RowMotionData. Those are quite abstract data that without the core idea of the game would not have any meaning.

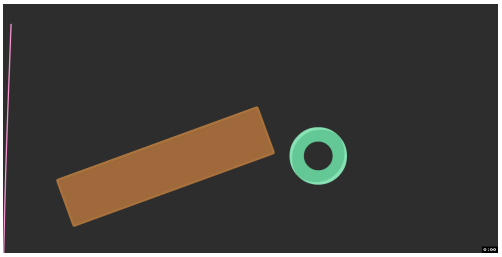
Quite relevant for the score management, so the purpose of our research is also the field *SessionData currentSession;* a field in which are stored global data about the overall experience.

The *GlobalManger* then is not the only one who defines the logics of the game. classes as Player and Fish, are used to represent the reality of our simulation, mainly a non playable character (the fish) and how fishing rod, within the simulation represented by just its dynamic wire.

Let's explore them better:

Player

The player is modeled in the application as the mere rope that is moved in the space by: fish movement, actual real movement of the physical fishing rod, and rotation of the wheel of the physical rod.



Tutorial on VelterRopes 2D in C# [19.]

Given these premises it is easy to understand our urge to find a way to recreate our own physical simulation engine.

We therefore came up with a strategy to render a physical rope to which are applied (at each game loop) different forces and positions of weak targets.

At each game loop the rope is subjected to:

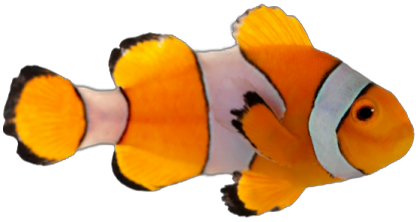
- *Gravity*: [**force**] applied to each node;
- *Rods Movements*: [**force**] applied as a directional vector (based on an accumulation of the acceleration x,y,z of the rod) to the first node of the rope, the origin, (in the top);
- *Keep the rope elastically anchored to a fixed point over the water*: [**weak target**] the target is the point origin, it need to resist the gravity but no the movements of the physical rod;
- *Collisions with the fish's head*: [**forces**] yes exactly, even if the case is not so resounding, when the fish collide with the rope the rope deflect. Applied as a force to the nodes with a magnitude as intense as is the fish head proximity.
- *The food is staying attached to the mouth for the bite duration*: [**weak target**] the last node of the rope (the one with the food) is targeted to the mouth of the fish for the bite time span with a smoothen window of intensity.
- *Cohesion of the rope*: [**forces**] to behave as a rope it is necessary to iterate multiple times to spread the tension forces of each node of the rope to all the rope.

The player is also responsible for:

- The damaging of the rope and notification of the rope-breaks event.
- The negotiation of the fish shift in condition in which the fish is hooked, (the shift from the desired delta pos at each cycle was a easy and fast solution to also apply trivial physics to the fish in case it need to be constrain in the movement by the rope).
- Bites of the fish, (a little mistake of responsibilities, that we let go).

Fish

The fish class model our non playable character. The fish is guided by an interpolated direction through the frames. And it is subjected to: a noise factor (to recreate a motion similar to the Brownian motion); an intentionality that should increase the probability that the direction would lead the fish toward the food; and a shift factor that should constrain the motion of the fish in case the fish is hooked.



4.4 Pure Data

The auditory feedback from pure data is managed with. Several different patches. They all listen for messages on different netreceive ports on localhost that are sent from pure data. When the patch *all_patches.pd* which contains all the other patches as subpatches they will be active as well and listen for commands.
The patches and their functions are the following:

- *play_sound.pd* listens for a message containing the filename of a WAV file and plays it one time. It is used for onetime feedbacks, like the bite of the fish, hooking of the fish, win & lose sounds.
- *play_music.pd* always plays four instrument tracks that together form a song simultaneously. The messages that it receives on the port set the volume of each individual instrument track. This indicates the intentionality of the fish during gameplay.
- *play_pitched_string_tension.pd* always plays a cosine wave of fixed frequency with a tremolo effect. The message it listens for will indicate the volume, the tremolo effect (the wavelength of the low-frequency oscillator multiplied with the input sine wave), and a boolean value if the sound should be played or not. This indicates if the wiretension is too high and the fish is about the rip the fishing string.
- *bubbles.pd* listens for an input message containing a string. If the string is one of four types the peak level, sustain level, attack time, decay time, sustain time, release time of the sound (which is coming from the liquid sound model *fluidflow~)*, bubbles per second and min and max bubble radius are set. The values of those sound parameters are fixed. If one of the four types comes in the connected parameters are set and the sound is played one time. This gives a feedback as to which movement the player just performed (to be exact which movement has been recognized by our system).
- *play_wheel_tick.pd* always plays the preset sound generated with the *dcmotor~* object that goes through a high pass filter. The messages that it receives indicate the speed and the volume of this sound. This tells the player how fast the wheel of the fishing rod is spinning.

All in all, it was important to us that all sounds are clearly distinguishable and audible. They give the player an exact feedback about the state of the game and do so in an intuitive and easily recognizable manner, which does not add a large cognitive load of memorizing all the sounds and their meaning.

4.5 Predictor

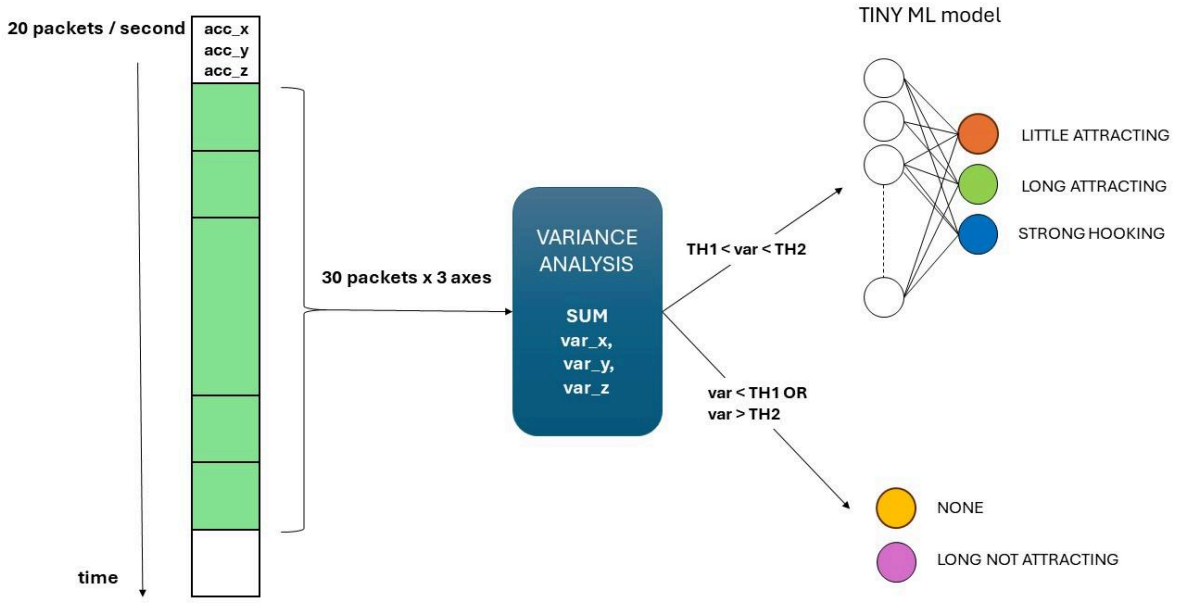
The predictor is design to be able to recognize a specific set of movements of the rod:

- NONE: no movement, fishing rod almost steady;
- LITTLE_ATTRACTING: small, short-lived spikes, as if there were a prey;
- LONG_ATTRACTING: large and sinuous movements, forming a figure “8”;
- LONG_NOT_ATTRACTING: too sudden movements;
- STRONG_HOOKING: strong spike to catch the fish.

To recognize these movements, we decided to rely on a Tiny Machine Learning (ML) model and on the analysis of the data related to the acceleration from the 3 axes accelerometer fixed on the top of the rod. This information is sended by the ESP32 via Wifi to the Processing instance running on the laptop. Ths system sets up a UDP connection which is faster than a TCP connection since it allows a very simple communication. The stream of data is then properly buffered and processed to filter the information. Indeed, data are previously analyzed based on the variance of the three accelerations to understand whether the user is making any movement or if he or she is doing a wrong shake of the rod (i.e. NONE, LONG NOT ATTRACTING).
Then, if the accelerations have a valid variance, the information is forwarded to the ML model which outputs the prediction of the kind of movement the user did (i.e STRONG HOOKING, LITTLE ATTRACTING, LONG ATTRACTING).

The neural network is designed and trained by exploiting Keras[12] and TensorFlow libraries and is composed of 2 hidden dense layers and 1 dense output layer which performs a softmax activation function to return each output neuron probability. The number of output neurons is equal to the number of valid rod gestures (reported in Chapter 3).
The model is trained by means of the Adam optimizer and the Cross Entropy loss function since we are dealing with a classification task. The training data have been collected from the rod and have been augmented to have much more samples since machine learning models are data hungry.

Relating to statistics, the model achieves good performances obtaining 99,9% of accuracy in training and 97,4% in test.



4.6 Hardware Manager

The program situated on the physical device (fishing rod) is developed in Arduino IDE, with the Arduino code (C-like), built with the compiler and uploaded through the uploader via microUSB ESP32-CAM-MB into the board ESP32-CAM. We decided to use this board because it is provided with a built-in module for the WiFi, and we need a wireless connection with the PC.

The program can be divided into 3 main parts:

- The configuration with some data structures and definition of utils functions;
- The setup phase in which devices synchronize and initialize themselves;
- The loop phase in which the program runs.

4.6.1 Configuration

We want to keep the software modular so that there are some configurations which help dynamic changes in the application. Those provided for this part of the system are:

- Network and WiFi (IPs, SSID, password);
- GPIO ports where various pins are connected;
- Protocol definition to send sensors data and to receive text to control actuators;
- The initialisation for the ADS1115 with its own address and ports of I2C;
- Utils function to manipulate strings;
- The callback function to receive UDP data.

4.6.2 Setup

The setup phase is needed to:

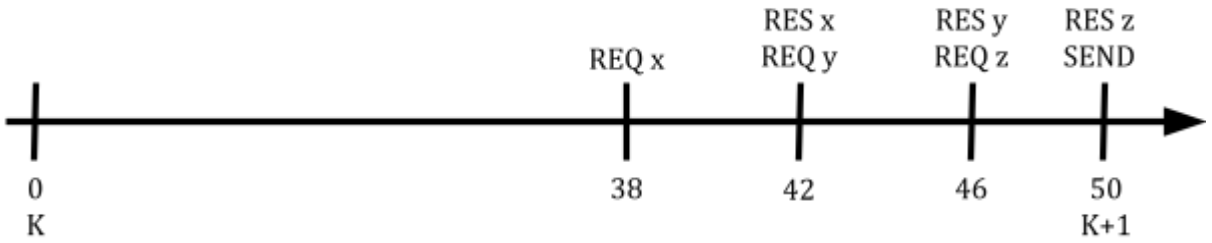
- Declare pins and direction (INPUT, OUTPUT, INPUT_PULLUP);
- Setup the ADS1115 and the accelerometer readings;
- WiFi connections and IP assignment;
- UDP Server and callback for the actuators setups.

Moreover, the setup phase uses the builtin red LED under the ESP32-CAM board (because we have no stable serial communication) to give information about the state in case of debug or malfunctions, explained in this table

LED behavior	Cause	Solution
LED blinks with short pulses	The ADS is not connected properly	Double check the Soft I2C pins connection (SDA, SCL)
LED blinks with regular pulses	The WiFi is not connected yet	Wait a few seconds, it may take some time (30-60 sec). Then check if the configuration for SSID and password is correct.
LED is fixed red	Error in connection or setting up the server	There is no need for a connection because of the UDP, but the IP of the PC must exist and the port must open. Check if the port is reachable

4.6.3 Loop

In the loop phase the main program for the hardware runs. It's divided into 2 time phases: the readings of the sensors and the computation and sending of data.
We need to read the state of one of the two phases of the rotary encoder every 500 microseconds and count every time the state changes (0 to 1, or 1 to 0); in the first change, we also check if the other phase is 1 or 0 to set the direction that obviously can change only once for each computation. Then every 200 milliseconds the computation is done, the velocity of the encoder is sent and the counter reset.
The ADS readings are requested, sent through I2C and sent to the PC about every 50 milliseconds: this is a good tradeoff between the sensibility in time of the accelerometer and the quantity of data that the predictor has to elaborate. The ADS unfortunately is not as immediate as a normal ADC reading: in fact, it uses I2C and even with the highest data rate, it takes 10/11 ms to read and send values; anyway, it's below the need threshold for our purposes of 50 ms.



We send raw values of the sensors to reduce computation since there is no reason to use standard unit measures (for example *revolutions per seconds* or *g*). For this reason, the initial idea was to implement the TinyML onto the board, but it requires a computation that goes beyond the potential of the ESP32-CAM, and the risk was to lose the real-time that we need for the rotary encoder.
The last part of the software is a callback function called when an UDP string is received: it checks if it respects standard protocol format, and retrieves a value to apply to ERMs.
There are also a few Python scripts to run on a PC to collect model samples and test the hardware: you can see them in the GitHub repository under the folder `MultisensoryFishingArduino`.

5. Evaluation

The hypothesis is that the full system with all three sensory stimuli (visual, auditory, haptic) let's users perform better and enjoy the gameplay more than a system with just any two of the three stimuli (visual-auditory, visual-haptic, auditory-haptic).

We did user testing to validate the system and to test the hypothesis. Each participant played the game four times, once with each of the above-mentioned settings. The game performance of the players were saved programmatically and they filled out a questionnaire after playing.

The independent variable is the setting of the game (which sensory stimuli are activated). The dependent variables are the number of won and the number of lost games, the time they needed for one round, the number of correct and the number of incorrect shakes they performed. The controlled variables will be the other game settings (e.g. how much the intenterest of the fish increases with each correct shake) that will be the same for all of the game rounds.

This was a within-subject experiment, since we let every participant play each of the four game modes we wanted to test. We did not compare the participants between each other.

We conducted user testing with 10 participants and we let each participant play the game four times:

1. once with every sensory feedback included (visual, auditory, haptic) *ALL*
2. once with no haptic feedback (only visual & auditory) *V-A*
3. once with no audio feedback (only visual & haptic) *V-H*
4. once with no haptic feedback (only auditory & haptic) *A-H*

5.1 Objective measures

We saved the participants performance in the game (play-time, attracting fish score, hooking fish score, retrieving fish score, damaged wire) to add an objective measure on how well they played.

The comparison shows the following:
Session duration: *V-A* had the lowest mean (127.8 sec), followed by *V-H* (152.4 sec), *ALL* (167.7 sec), and *A-H* (171.0 sec). The fastest overall game was played in *V-A* in 48.9 sec.
Attracting fish score: *V-H* had the highest mean (2.29), followed by *ALL* (1.02), *V-A* (0.70), and *A-H* (-1.86). The highest overall score was played in *V-H* with 7.86.
Hooking fish score: *V-A* had the highest mean (-1.18), followed by *V-A* (-2.61), *A-H* (-2.74), and *ALL* (-19.76). The highest overall score was played in *V-H* with 3.27.
Retrieving fish score: *V-H* had the highest mean (0.76), followed by *ALL* (0.40), *V-A* (0.15), and *A-H* (-1.64). The highest overall score was played in *ALL* with 3.71.
Damaged Wire: *V-H* had the lowest mean (0.43), followed by *ALL* (0.51), *A-H* (0.54), and *V-A* (0.67). The lowest overall damage taken in one game was played in *A-H* with 0.

We do not need to perform hypothesis testing since we see that in *ALL,* participants did not perform consistently better than in the other combinations of senses.

We explain these results with the setting of our user tests. Since we let all users play the game mode *ALL* as their first game and then all of the others in order, they performed mostly worst on their first game and improved with each additional game. The effect of the presence/missing of the sensory stimuli was largely overshadowed by this effect of learning. If this explanation holds any ground, in a future user study we have to let the users play multiple games to learn the gameplay before we start the actual games that evaluate their performance with the different combinations of senses.

5.2 Questionnaire

After the participant played, we asked them to fill out a questionnaire, for every one of the four game modes, where they had to answer their agreement with the following three statements (on a scale of 1-5 respectively):

1. The game with these settings was fun to play.
2. It was easy to catch the fish in these settings.
3. The combination of sensory stimuli helped me in knowing what I had to do.

We compared the responses for *ALL* with each of the responses for the other modes and examined them for statistical significance. Let's show an example for the comparison *ALL vs. V-A* and the question about the *fun*:
The mean response for *fun* with *ALL* was 4.7 with a standard deviation of 0.48 while the mean response for *fun* with *V-A* was 4.3 with a standard deviation of 0.83. The degrees of freedom are 18 (10 responses + 10 responses - 2).
Our null hypothesis is that there is no difference between the *fun* for the two game modes and we want to show that with a p-value of 0.05 in a one-tailed test (since we see it as given that *ALL* is the most fun game mode). From the means and standard deviation, we get a calculated t-value of 0.968. We look up the critical p-value in a t-table with our values (0.05, one-tail, 18 df) which is 1.734. The calculated p-value is therefore smaller than the critical value which means we accept the null hypothesis.

This procedure was repeated for all nine comparisons and here are the results:

fun (*ALL* mean: 4.7, std. dev: 0.483)

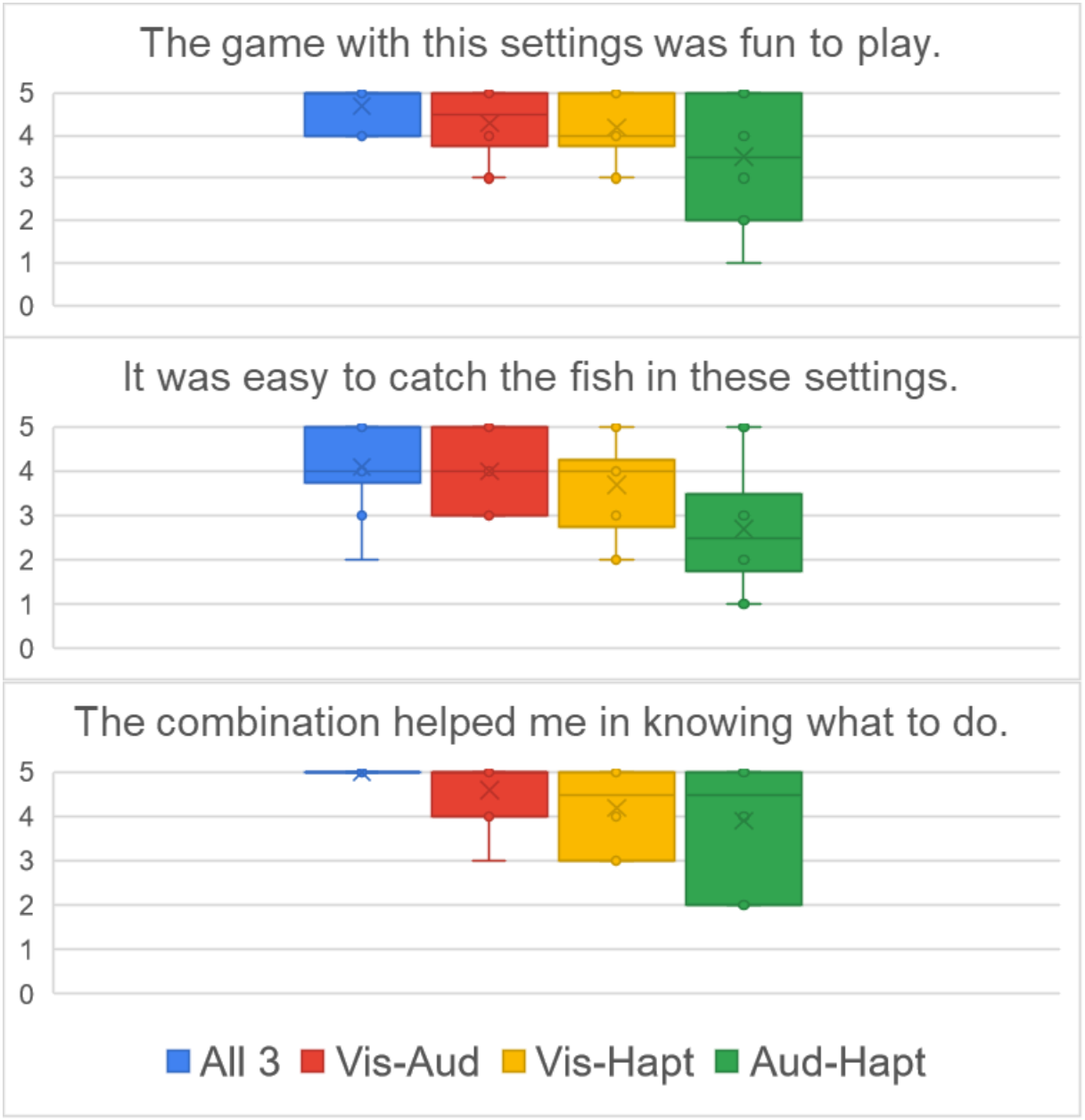
- *V-A* (mean: 4.3, std. dev: 0.823) t-value: 0.968
  We do not have enough evidence to say that *ALL* is more fun than only *V-A*
- *V-H* (mean 4.2, std. dev: 0.789) t-value: 1.243
  We do not have enough evidence to say that *ALL* is more fun than only *V-H*
- *A-H* (mean: 3.5, std. dev: 1.509) t-value: 1.905
  We have enough evidence to say that *ALL* is more fun than only *A-H*

ease (*ALL* mean: 4.1, std. dev: 0.994)

- *V-A* (mean: 4.0, std. dev: 0.816) t-value: 1.746
  We have enough evidence to say that with *ALL* it is easier than only *V-A*
- *V-H* (mean 3.7, std. dev: 1.059) t-value: 1.616
  We do not have enough evidence to say that with *ALL* it is easier than only *V-H*
- *A-H* (mean: 2.7, std. dev: 1.418) t-value: 1.835
  We have enough evidence to say that with *ALL* it is easier than only *A-H*

help (*ALL* mean: 5.0, std. dev: 0.0)

- *V-A* (mean: 4.6, std. dev: 0.700) t-value: 1.809
  We have enough evidence to say that *ALL* helped more than only *V-A*
- *V-H* (mean 4.2, std. dev: 0.919) t-value: 2.753
  We have enough evidence to say that *ALL* helped more than only *V-H*
- *A-H* (mean: 3.9, std. dev: 1.370) t-value: 2.538
  We have enough evidence to say that *ALL* helped more than only *A-H*



Box-plots showing the distribution of the answer scores for the different game modes

6. Discussion and conclusions

The development of the project was beneficial for learning purposes, because it's a real application that involves multiple senses. For each sense, there is a different framework, with a different language and various challenges. For example, we treated both hardware and software, for haptics, videos, and sounds; we need knowledge of different kinds of disciplines: from electronics skills to network communication; from programming abilities to gamification; these are very important to build a multisensory (but not only) device.

There are some things that the experience in developing this project has taught us: for example, the usage of the ESP32-CAM was a bad choice, because it's built mainly to record with the camera, so many difficulties arose when we tried to implement basic features; the normal ESP32 would be a better choice.

Even if the project involves 3 different senses (sight, hearing and touch), your perception is not completely involved in the system as in a real situation, but the simulation works quite well. To improve this thing you need, for instance, to use an AR or better VR. This could be a future work, to make the device effectively marketable. Other improvements could be to use strong models to detect additional and non-essential movements or implement other features and rules in the game manager, making it more attractive.

The objective measures that we made on the game performances of the participants seemed a little bit randomly distributed over the different games. As mentioned in the previous section, we attribute this largely to the effect of learning that the participants showed over the course of their four games. Even though the last three games had one sense fewer than the first one while everything else remained the same, they did not show their best performance in the first game they had. This could be because they were still getting to know the game in this first play since it was their first ever experience with the fishing rod and the game.

Considering this, it is surprising to see, that the first game with *ALL* was not the worst of the four games across the board but mostly in the second or third position. While the game mode that did score worst in several of the scoring criteria was interestingly *A-H* which was always played last and therefore should have gotten the largest benefit of the learning effect. It seems that the missing visual sense in this game mode had a larger effect on the scores than the fact that participants played the game for the fourth time.

The results from our user study were very interesting. The participants' fun that they reported was large in all of the game modes. But when comparing the game mode *ALL* with the rest, we can say with statistical significance the participants reported having less fun without the visual sense than with all three senses present. We explain this with the fact that humans are inherently visual creatures and the sense of fun can be largely influenced by what is in a human's visual perspective. In the case of *A-H*, the participants were staring at a blank screen while playing.

The question "It was easy to catch the fish in these settings." showed a significant difference between *ALL* and *V-A* & *A-H*. For the combination *V-H* we can not show with statistical significance that the respondents found it easier to catch the fish with all 3 senses than with the auditory sense missing. But still, for two of the senses (haptic & visual) the results suggest that catching the fish (so achieving the goal of the game) is easier with all three senses than it is when one of them is missing.

For the question "The combination of sensory stimuli helped me in knowing what I had to do.", a statistical significance showed up in every single one of the comparisons. This shows that using all three senses gives you a lot more help in the form of information on what to do than any combination of two senses respectively. The numbers to this question suggest that visual is the most helpful sense for our game, while auditive the second, and haptic is the third most helpful. But there was still a statistical significance between *ALL* and *V-A* even though "only" the least informative of the three senses was missing.

In conclusion, even though the senses in our game largely overlap and describe the same information about the game's state, users found they got more help with the gameplay when they were given all three senses than any combination of senses where one of them was missing.

7. Group members contributions

Jan

- designed & printed the handle
    - design the model on Shapr3D, slice with Ultimaker Cura;
    - 3D print the handle of the reel (in collaboration with Manuel).

- Auditory feedback with pure data:
    - search WAV files, create PD sketches;
    - send messages to PD in appropriate game events (created by Riccardo) in Processing.

- User-testing and evaluation:
    - wrote the questionnaire in English and Italian;
    - design of the test questions and hypothesis, calulation of statistical significance;
    - In collaboration with the group: test and evaluation sessions with participants.

Manuel

- Mainly the hardware and software for hardware part:
    - Design of the device and electronics;
    - Printed with 3D printer the rod (in collaboration with Jan);
    - Connected, fixed and soldered components in different boards;
    - Developed entire embedded software in Arduino IDE;
    - Developed a communication utility with the rest of the software (in collaboration with Marco);
    - In collaboration with the group: session to test parts together, fix and finetune movements.

- Software for computer:
    - Some software to test various components and communication;
    - Software to collect samples for the model;
    - Collecting samples for the model to predict events;
    - Data analysis of the variance on prediction software: implementation of an algorithm to recognize none and not attracting.

- Test & evaluation:
    - Script for the tutorial for testing, with metrics, feedback, actions and how to play;
    - Graphics for the poster to attract people to play;
    - Information sheet in italian and english with appendix with expected results, feedbacks and actions;
    - In collaboration with the group: test and evaluation sessions with participants.

Marco

- Design and construction of the prototype of the rod:
    - Welding of wires and components
    - Laying of wires
    - Building of the support for the ESP32 and the fishing reel (designed and printed by Jan).

- Communication ESP32 - Computer - TinyML model
    - Software to manage the communication between Processing (running on the laptop), and the ESP32 fixed on the rod. In particular receive/send data from sensors/vibrators fixed on the rod (in collaboration with Manuel).

    - Software to send/receive data to the TinyML model running on a python instance.

- Design and training of the Tiny ML model
    - Software for augmenting data to get more samples (collected by Manuel) for the model training
    - Software for processing data to be feeded to the ML model
    - Software for training and test the model

Riccardo

- Hardware:
    - Design of the device: rotary encoder joint (collab. Marco), small changing in the appearance;

- Software:
    - Python script (*Main.py*) to listen to the stream of data coming from OpenFace2.0 face recognition sub-processes.
    - Refinements of the predictor Python script *tinyML.py* (collab. Marco, Manuel).
    - Processing code: I wrote all the files except *CallPD.pde*, *AudioSensoryModule.pde*, *SensoryInputModule.pde* and *HapticSensoryModule.pde.*
    - Collaborate to write the PureData files: *bubbles.pd*, *play_pitched_string_tension.pd* and *play_wheel_tick.pd*.

- Other:
    - Class Diagram of the processing code following the protocol UML 2.5
    - Scripting for the video presentation
    - Movie making (with the team as actors) and montage of the video
    - I wrote Batch files to run faster the scripts, README.md

- Test & evaluation:
    - Debug utility, and a utility that deploy a presentation of all game events in FileSystem.
    - Fine tuning of the game parameters as well the one for Pure Data (in collab. Jan), haptic sensors (in collab. Marco and Manuel) and visual module.
    - Test with some users

8. References

1. GitHub full project, https://github.com/RiccardoCalcagno/MultisensoryFishingRod.git
2. ESP32-CAM features and port diagram, https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/DFR0602_Web.pdf
3. ESP32-CAM other useful information, https://randomnerdtutorials.com/esp32-cam-ai-thinker-pinout/
4. ADS1115 datasheet, https://www.ti.com/lit/ds/symlink/ads1115.pdf?ts=1704867448825&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FADS1115
5. ADS1115 library, https://www.arduino.cc/reference/en/libraries/ads1x15/
6. Soft I2C library, https://www.arduino.cc/reference/en/libraries/softwire/
7. MOSFET controlled with EMI suppression circuit, https://www.precisionmicrodrives.com/how-to-drive-a-vibration-motor-with-arduino-and-genuino
8. Rotary encoder datasheet, https://docs.rs-online.com/6487/0900766b813ecfbe.pdf
9. Tiny Machine Learning, https://github.com/jaredmaks/tinyml-on-the-edge
10. UDP communication in Java, https://www.baeldung.com/udp-in-java
11. UDP communication in Python, https://wiki.python.org/moin/UdpCommunication
12. Keras API, https://keras.io/api/
13. Wii Play, https://it.wikipedia.org/wiki/Wii_Play
14. Parchment Skin Illusion, https://youtu.be/Ca3N-ih-sCQ
15. OpenFace2.0 open source Project, https://github.com/TadasBaltrusaitis/OpenFace
16. Guide on how to boost the PVector processing class to implement Looking-at-point functionality: https://copyprogramming.com/howto/how-to-rotate-a-vector-by-a-given-direction?utm_content=cmp-true
17. Software that we used to create virtual local ports COM1, COM2 in window11: https://www.hhdsoftware.com/virtual-serial-ports
18. Pure Data: https://puredata.info
19. Tutorial on how to realize a Verlet Rope in 2D in Unity: https://toqoz.fyi/game-rope.html

9. Appendix

9.1. 3D environment and rotations in processing

A taste of the level of complexity to handle rotations in 3D with processing. Thanks to our custom class *Vec3* [16.] (extending *PVector*) we were able to use a powerful function: *lookAt(...)* and avoid that the fish would be rendered upside down.

```
var fishRotation = fish.getFishRotation();
var fishPose = fish.getPos();

Vec3 powerdRotation = new Vec3(1,0,0);
PVector rotationEnlarged = fishRotation.copy().setMag(100);
PMatrix3D rotationMatrix = powerdRotation.lookAt(rotationEnlarged, new PVector(0, -1, 0));
PVector targetGround = new PVector();
rotationMatrix.mult(powerdRotation, targetGround);

pushMatrix();
scale(scaleScene);

translate(width/2 + fishPose.x, height/2 +fishPose.y - 10, fishPose.z);
applyMatrix(rotationMatrix);
rotateY(-PI/2);
rotateX(PI -targetGround.x);
```

The code allows the image of the fish to be rendered well oriented

9.2. A dynamic rope subject to physics

After watching a tutorial on how to create a 2D rope subject to physics in C# [19.] We tried to create from scratch our 3D version of it in Processing. With the addition not only to push the rope applying a force vector but also to target the position of a certain node to a certain point of space.

```
void update(){
  applyForcesOfRod();

  if(gameManager.isFishHooked() == false){
    handleCollision();
  }

  applyPhisics();

  // Lazy application of all the forces collected during the cicle
  for (VerletNode node : nodes) {
    node.applyCachedForces();
  }

  float rigidityOfRod = totalNodes * 2;
  // More iterations define the rigidity of the rope
  for(int i=0; i<rigidityOfRod; i++){
    for (int j = 0; j < nodes.length; j++) {
      nodes[j].applyConstrains((j< nodes.length-1)? nodes[j + 1]: null, nodeDistance,

      if( i ==  rigidityOfRod-1){
        nodes[j].position.x = lerp(nodes[j].position.x, 0, 0.003);
        nodes[j].position.z = lerp(nodes[j].position.z, 0, 0.003);
      }
    }
  }
```

update of the player (our rope), by taking care of:
collision with fish, gravity, physical constraints and user pulling the fishing rod

9.3. Abstract Sensory Output Modules

In the previous paragraph regarding the output modules (*4.3.2. Output*) we explained why it was so important to define an abstract class that would be implemented by all the sensory output modules.

In the following code snippets it is shown the contract of this abstract class, with all the public resources that can be exploited by the specialization classes and the methods to be overwritten in order to attach to the flow of the events of the application.

```
interface PublicFish{
  float fishWidth = 700, fishHeight = 183;
  // return a vector 3 of the position of the fish in the acquarium
  PVector getPos();

  PVector getDeltaPos();

  // Fish intentionality is a coefficent from 0 to 1 that tell how much willing is the
  // fish to bite the hook... 0 => random movement, 1 => strait movement to the hook.
  float getIntentionality();

  PVector getFishRotation();
}


interface OutputModulesManager{
  // true <=> the player is in the second part of the session (retreiving the hooked fi
  boolean isFishHooked();

  int getSizeOfAcquarium();

  // Use it if you need to have some information about the fish
  PublicFish getFish();

  VerletNode[] getNodesOfWire();

  PVector getCameraPosition();

  ShakeDimention getCurrentShake();

  DebugUtility GetDebugUtility();
}
```

Interface of all the output modules
With the resources that they can exploit
(as outputModuleManager implemented
by the gameManager, PublicFish)

```
abstract class AbstSensoryOutModule{

  OutputModulesManager outputModulesManager;
  AbstSensoryOutModule(OutputModulesManager _outputModulesManager){
    outputModulesManager = _outputModulesManager;
  }

  // Once per gameLoop
  void OnRodStatusReading(RodStatusData dataSnapshot){}

  void ResetGame(){}

  // Asyncronous meningful events
  void OnShakeOfRod(ShakeDimention rodShakeType){}

  // event fired when the fish is touching the hook.
  // 1 event of tasting the bait has at least 0.8 sec of distance between each others
  void OnFishTasteBait(){}

  void OnFishHooked(){}

  void OnFishLost(){}
  void OnFishCaught(){}
  void OnWireEndedWithNoFish(){}

  void OnEndGame(){}
}


class RodStatusData{

  // from -1 to 1. Normalized
  // negative speed for retreiving the wire, positive velocities for relising wire.
  float speedOfWireRetrieving;

  // From 0 to 1.... 0 means not in tension. if isFishHooked == true => tention = 0
  // tension is biggest when the fish is pulling in the opposite direction of the wire
  // and the speedOfWireRetrieving is equal to 1
  float coefficentOfWireTension;

  // Check RawMotionData in SensoryInputModule script
  RawMotionData rawMotionData;
}
```

```
for(int i=0; i<3; i++){
  if(_playerInfo.selectedModalities[i]){
    AbstSensoryOutModule moduleToAdd = null;
    switch (i) {
      case 0:
        moduleToAdd = new VisualSensoryModule(this);
        break;
      case 1:
        moduleToAdd = new AudioSensoryModule(this);
        break;
      case 2:
        moduleToAdd = new HapticSensoryModule(this);
        break;
    }
    sensoryModules.add(moduleToAdd);
  }
}
```

This code snippet inside the function
StartGameWithSettings(..) in the GameManager
instantiate the exact module that needs to be
updated as the user chose in the settings

9.4. Game loop, a relevant insight of the app functioning

The following code might be useful to have an overview on the main events happening at each cycle of our application. It is shown the point in which each output module receives its fresh data for the current frame.

The code also highlights how our system intrinsically require an elaboration of the input that goes beyond a mathematical transformation function. Player class (with complex physics simulations) and Fish Class (with the introduction of random noise for its movement) define a much more complex interaction.

Ex. as reported in the code… retrieving the wire by activating the rotary encoder will produce a different RoadStatusData data. Something that will cooperate with the random movement of the fish and the shape of the rope to produce a different coefficient of tension of the wire => perceivable in the output.

```
void gameLoop(){

  updateState();

  hint(ENABLE_DEPTH_SORT);

    if(haloForShakeRodEvent > 0){
      currentRodState = cachedShakeRodEvent;
      for (AbstSensoryOutModule sensoryModule : sensoryModules) {
        if(prevCachedShakeRodEvent != ShakeDimention.NONE && cachedShakeRodEvent != ShakeDimention.NONE){
          sensoryModule.OnShakeOfRod(ShakeDimention.NONE);
        }
        sensoryModule.OnShakeOfRod(cachedShakeRodEvent);
      }
      haloForShakeRodEvent = 0;

      if(currentState == GameState.AttractingFish && currentRodState == ShakeDimention.STRONG_HOOKING){
        if(player.HasHookedTheFish()){
           setState(GameState.FishHooked);
        }
      }
    }

    fish.UpdateIntentionality(currentRodState);

    RodStatusData data = calculateRodStatusData();

    player.TornWireOnRodMovments(data.rawMotionData);

    player.update();

    for (AbstSensoryOutModule sensoryModule : sensoryModules) {

      sensoryModule.OnRodStatusReading(data);
    }


  hint(DISABLE_DEPTH_SORT);
}
```

```
private RodStatusData calculateRodStatusData(){

    if(haloForWireRetrieving <= 0){
      cachedSpeedOfWireRetrieving = 0;
    }
    if(haloForRawMovements <= 0){
      cachedRawMotionData = new RawMotionData();
    }

    RodStatusData newData = new RodStatusData();

    newData.speedOfWireRetrieving = cachedSpeedOfWireRetrieving;
    newData.rawMotionData = cachedRawMotionData;

    newData.coefficentOfWireTension =
      player.UpdateWireRetreival(newData.speedOfWireRetrieving);

    haloForWireRetrieving--;
    haloForRawMovements--;

    return newData;
}
```

Portion of game loop with a zoom-in in the calculateRodStatusData function

9.5. Fine-tunable parameters

In order to develop the application in a more modular way we chose to divide our tasks and work simultaneously on the Processing code and on the fishing rod. The code was then designed in a way to reduce as possible changes based on peculiarity of the sensors, such as sensitivity, range, flickering, distribution ecc.. The code synthesized these variability into these parameters. Fine-tuned in the last testing phase all together.

```
// coefficent expressing how much the valence of a certain
// shake change the intentionality of the fish at each cicle
float intensityOfValenceOfShakesForAttraction = 0.001;

// Speed of the fish on free condition
float stepSize = 2;

// Speed of the fish when hooked
float stepSizeWhenHooked = 4;

// threshold distance necessary to shape the speed of the
// fish in it environment, with food vision
float distanceFromFoodWhenStartToDecellerate = 50;

// reaching this value with the counter:
// timeSinceAttractionWasPositive define the maximum extream
// of intentionality unlocked. [Needs to be a multiple of 5]
int strinkingTimeToReachOptimumAttractability = 500;

// If no shake is introduced since this ammount of frames,
// the intentionality of the fish start reaching the 0 with
// a certain speed (Forgetting)
int numFramesAfterFishstartForgetting = 300;

// Speed of forgetting his intentionality
float speedOfForgettingIntentionality = 0.002;

// this express a value of intentionality to reach the bottom
// of the see when the fish is hooked
float intentionToGoDownWhenHooked = 0.24;

// set this values to adjust the degree in wich a certain
// shake is attracting - scaring the fish , intentionality
// can go from -0.3 to 0.8
public void setShakeAttractionMapping(){
  shakesValenceForAttracting.set("NONE", 0);
  shakesValenceForAttracting.set("LITTLE_ATTRACTING", 0.5);
  shakesValenceForAttracting.set("LONG_ATTRACTING", 1);
  shakesValenceForAttracting.set("STRONG_HOOKING", -5);
  shakesValenceForAttracting.set("STRONG_NOT_HOOKING", -4);
}
```

Parameters to control the Fish

```
// max damage supported by the wire
float maxDamage = 50;

// Bigger => More the wire should bounce in the moment
// in wich the fish touch it
float multipliyerOfCollisionReaction = 3;

// Gravitational force in 3D
PVector weightOfNode = new PVector(0, 0.8, 0);
PVector weightOfBait = new PVector(0, 48, 0);

// This should define the lenght of the rope, not the
// position of the hook, this would stay evrytime almost
// in the middel of the boundingBox
int totalNodes = 100; // 200

// This is a constant force that is appplied to the origin
// of the wire (top extream of it) in order to make it
// reace a default position if the user is not applying
// other forces by moving the phiscal rod
float speedToReachTheIdleOrigin = 0.1;

// this is a coefficent to be fineTuned in order to
// intensify or decrese the force of the movements of the
// phisical rod, need to be adjusted in order not to make
// the hook be throunw out of the water
float intensityOfRodMovments = 10000;

// if the mouth of the fish is more distant form the food
// than minDistanceOfMouthFromFoodToHook at the moment of
// the Strong_Hooking shake than the hook do not success.
int minDistanceOfMouthFromFoodToHook = 20;

// Average the periods in which the fish is pushing the wire
// it is repeditly biting (just a taste actually) the hook,
// but with a minimum delay between each bites, namely this
// value. Expect a variance of +- 0.2*numLoopsBetweenBites
// This value increase with the intentionality of the fish
// map(intentionality, 0, 0.8, max, min)
int maxNumLoopsBetweenBites = 120;
int minNumLoopsBetweenBites = 80;

// Dial this one to require to the user more or less
// reactivness to the fish tasting
// they start from this value and each time the user fail
// to strinke increase of 4 until numLoopsForReactivness * 3
int numLoopsForReactivness = 17;

// quantity of wire retreived at each step
float maxSpeedOfWireRetreiving = 2;

// when the wire is idle which is the offset of the bait
// from the center of the room? (it should be a little bit
// below so that there is more wire to be retreived)
float YoffsetOfBaitFromCenterOfRoom = 30;
```

Parameters to control the Fish

9.6. Thread-safety haptic patterns

In the following code snippet is shown our strategy to send the appropriate value to the servo motors at a certain moment. This design takes in consideration the possibility that multiple game events that require a certain haptic sequence might overlap. Therefore, it was necessary to take in consideration priorities to allow the override of a previous pattern. It was also necessary to keep the access of thread to the states of the events and pattern safe, the solution proposes a simple invocation of *InputSensoryModule.getCurrentValue()* that univocally provides the right value to the thread.

```
private int[][] getPattern(FishingEvent event){

  switch(event){

    case ROD_READ:
      float tens = lastTension;
      int value = int(map(lastTension, 0.1, 1, MIN_VIBRATOR_VALUE, MAX_VIBRATOR_VALUE));
      return new int[][] { {255, 20}, {value, -1} };

    case FISH_TASTE_BAIT:
      return new int[][] { {MAX_VIBRATOR_VALUE, 100}, { 0, -1 } };

    case FISH_LOST:
    case WIRE_ENDED:
      return new int[][] { {HALF_VIBRATOR_VALUE, 500}, { 0, 500 }, { HALF_VIBRATOR_VALUE, 500 }, { 0, 500}, {HALF_VIBRATOR

    case FISH_HOOKED:
      return new int[][] { {MAX_VIBRATOR_VALUE, 200}, { HALF_VIBRATOR_VALUE, 100 }, {MAX_VIBRATOR_VALUE, 100}, { HALF_VIBR

    case FISH_CAUGHT:
      return new int[][] { {HALF_VIBRATOR_VALUE, 200}, { 0, 180 }, { HALF_VIBRATOR_VALUE, 200 }, { 0, 180}, {HALF_VIBRATOR

    case NONE:
    case END:
    default:
      return new int[][]{ { 0 , -1} };
   }
}
```

getPattern(..) used to describe the kind of pattern to perform for each haptic event

```
public int getCurrentValue(){
  if(millisecSinceEvent == null){
    millisecSinceEvent = millis();
  }
  var patternChosed = getPattern(event);
  var index = ReadCurrentIndexOfPattern
                (patternChosed, getMillisFromEvent());

  int outValue = -1;
  if(index>=0){
    outValue = patternChosed[index][0];
  }
  return outValue;
}
```

getCurrentValue(..)

```
priorities.put(FishingEvent.NONE, 1);
priorities.put(FishingEvent.ROD_READ, 1);
priorities.put(FishingEvent.FISH_TASTE_BAIT, 2);
priorities.put(FishingEvent.FISH_HOOKED, 4);
priorities.put(FishingEvent.FISH_LOST, 4);
priorities.put(FishingEvent.FISH_CAUGHT, 4);
priorities.put(FishingEvent.WIRE_ENDED, 4);
priorities.put(FishingEvent.END, 10);
```

The hashMap that contain the priority for each haptic event

9.7. Efficient debug behaviors and Informative debug messages

With the purpose to speed up and simplify the last phase of testing and fine-tuning of the system was also design a debug utility with the power to:

1. Allow the programmer to activate/deactivate a list of behaviors useful in a debug situation.
2. Write in a file `MultisensoryFishingProcessing/data/DebugGames.txt` a comprehensive note containing all the game events of a session necessary to calculate the final score, so that, after colleging a good amount of average plays (20) we were able to define for each event a mean and a variance and in the end came up with a balanced summative score.

```
void setup() {
  background(99, 178, 240);
  //fullScreen(P3D);
  size(1400, 1400, P3D);
  hint(DISABLE_OPENGL_ERRORS);

  HashMap<DebugType, Boolean> debugLevel = new HashMap<DebugType, Boolean>();
  debugLevel.put(DebugType.IOFile, true);
  debugLevel.put(DebugType.StartAlreadyWithFishHoked, false);
  debugLevel.put(DebugType.FishMovement, true);
  debugLevel.put(DebugType.InputAsKeyboard, false);
  debugLevel.put(DebugType.ConsoleAll, true);
  debugLevel.put(DebugType.ConsoleAlowFrequent, false);
  debugLevel.put(DebugType.ConsoleIntentionAndTension, false);
  debugLevel.put(DebugType.ConsoleAlowRawRodInputs, false);
```

Debug behaviors to be activated / deactivated in the setup() function

Example of message printed in the DebugGames.txt. Each paragraph represents a kind of Game Event and the lines beneath represent a timeline for the entire duration of the experience.

9.8. Shakes recognition through ML and analysis of accelerations

```python
while True:
    dataReceived = receiveData()

    # Filling the circular Buffers
    if(len(dataReceived) > 0):
        buffer += dataReceived
        buffer_for_strong_hook += dataReceived[1]

    # Checking if the short buffer for STRONG_HOOKING has data
    # If there is a spike on y axis big enough => send STRONG_HHOKING
    if(len(buffer_for_strong_hook) == BUFFER_MAX_SIZE_STRONG_HOOK):
        Y = np.mean(buffer_for_strong_hook)
        if(y < 0.05 and ((time.time() - time_last_strong_hook) > MIN_DELAY_BETWEEN_STRONG_HOOKS)):
            sendPrediction(2)    # 2 is the code for STRONG_HOOKING
            time_last_strong_hook = time.time()
        del buffer_for_strong_hook[:1]

    if(len(buffer) == BUFFER_MAX_SIZE):
      del buffer[:WINDOW_SIZE] # remove the oldest data from the buffer
      inputs = np.array(buffer)

      # variance = var over all window, early_variance = var orver the last bit of window
      variance, early_variance = calculateVariances(inputs)

      if early_variance < 0.003:
        prediction = 3
      else:

        # Predict the shake with MACHINE LEARNING
        inputs_ml = inputs[None, :] # add the batch dimension
        output = model.predict(inputs_ml)

        # if the prediction is less then a certain value of probability, the event is considered as NONE
        if np.max(output) < 0.7:
          prediction = 3 # NONE
        else:
          prediction = np.argmax(output)
          if prediction == 1 and variance > 0.1:
            prediction = 4 # long_NOT_attracting
          if prediction == 2:
            prediction = 0
```

Commented script in python that shows the shakes recognition strategy

9.9. NPC (Fish) movement

In the project was very important to design a simple intelligence for the fish that could be able to integrate the need for a believable *brownian* motion (random in the 3D space) but also a movement that can be directed towards a target (the bait) with more or less intention (-0.3, 0.8).

The code below shows how we reached this necessity. *CalculateDeltaTarget(..)* allow the Fish class to direct smoothly the fish towards its dynamic trajectory being influenced by its intentionality, space constrains and the presence of the wire in case the fish is hooked.

```
void UpdatePosition() {

  float speed = 0;

  if(gameManager.debugUtility.debugLevels.get(DebugType.FishMovement) == true){
    direction =|calculateDeltaTarget();
    speed = adjustSpeed();
  }

  direction.setMag(speed);

  fishShift = player.NegotiateFishShift(direction);

  pos.add(PVector.add(direction, fishShift));

  // Constrain fish within the cube
  pos.x = constrain(pos.x, -boxsize/2, boxsize/2);
  if(player.countDownForCapturingAnimation == -1){
    pos.y = constrain(pos.y, -boxsize/2, boxsize/2);
  }
  pos.z = constrain(pos.z, -boxsize/2, boxsize/2);

  //DebugHeadColliderAndSpeedVector();
}
```

Function to update direction and position of the fish

```
PVector calculateDeltaTarget(){
  float targetIntentionality = intentionality;
  PVector deltaFood = new PVector();
  if(gameManager.isFishHooked() == false){
    calculateDeltaFood(deltaFood);
  }
  else{
    deltaFood = new PVector(0,1,0); // tendency to stay in the bottom during the capturing
    var intentionToGoDown = intentionToGoDownWhenHooked;
    var time = (frameCount - gameManager.currentSession.startTime) / frameRate;
    if(time > 150 && time < 200){
      intentionToGoDown = map(time, 150, 200, intentionToGoDown, 0);
    }
    targetIntentionality = (pos.y < 0) ? intentionToGoDownWhenHooked:
                              map(pos.y, boxsize/2, 0, 0, intentionToGoDownWhenHooked);
  }

  // Update fish's position based on intention
  float noiseScale = 0.01;
  float noiseX = map(noise(frameCount * noiseScale), 0, 1, -0.1, 0.1);
  float noiseY = map(noise((frameCount + 1000) * noiseScale), 0, 1, -0.1, 0.1);
  float noiseZ = map(noise((frameCount + 2000) * noiseScale), 0, 1, -0.1, 0.1);
  float noisedistance = sqrt(noiseX*noiseX + noiseY*noiseY + noiseZ*noiseZ);

  noiseX /= noisedistance;
  noiseY /= noisedistance;
  noiseZ /= noisedistance;

  // Calculate the weighted combination of random movement and intentional movement

  if(targetIntentionality > 0){
    return new PVector(
      lerp(noiseX, deltaFood.x, targetIntentionality),
      lerp(noiseY, deltaFood.y, targetIntentionality),
      lerp(noiseZ, deltaFood.z, targetIntentionality)
      );
  }
  else{
    return new PVector(
      lerp(noiseX, -deltaFood.x, abs(targetIntentionality)),
      lerp(noiseY, -deltaFood.y, abs(targetIntentionality)),
      lerp(noiseZ, -deltaFood.z, abs(targetIntentionality))
      );
  }

}
```

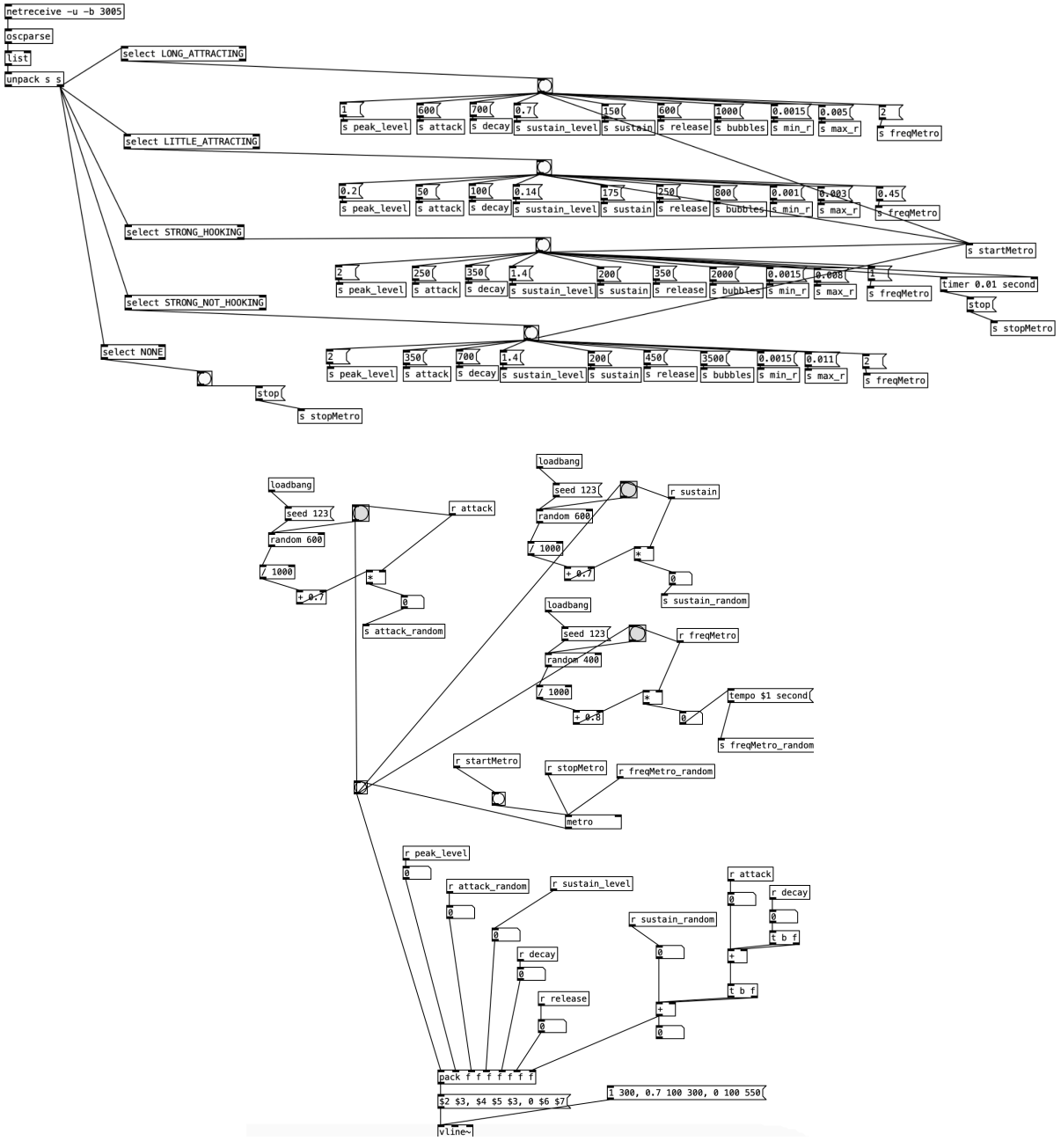
Function to calculate the next direction based on intentionality

9.10. Developing with Pure Data

After a long training period used to explore the libraries and functionality of Pure Data and master its functioning. We came up with interesting scripts that we guess might simulate sufficiently realistic fishing sounds.

One interesting sketch that we came up with was *bubble.pd*. In the following code snippets it will be possible to see:

1. A reuse of the code provide in the library STD for the simulation of the water's movement;
2. How we used the notion taught by our professor about the shape of a natural sound (attack, decay, sustain, release) to manipulate the sound provided by the library in a singular clear movement of the water, triggerable when defined by the network communication;
3. A good use of the notion of the objects metro and random to variate the length and the shape of each water movement in order to create a sequence of movement that will not sound as a boring cyclical reproduction of the same sound;



9.11. Brochure

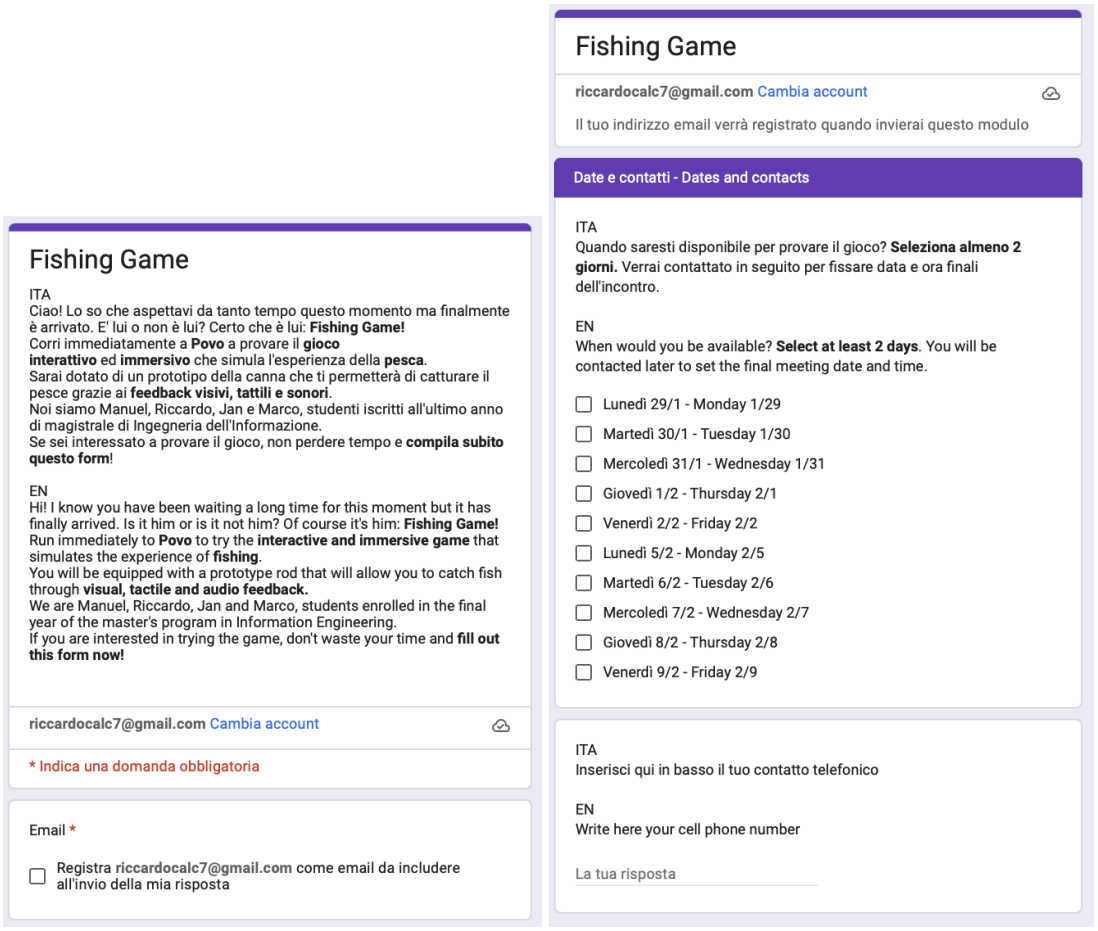
We used this infographic to attract and inform students of the *unitn* to participate in our experimentation.

Brochure of the experimentation

9.12. Appointment Scheduling Form

We then attached the link of the form to schedule the appointments for experimentation with the brochure at the email

Link to the form
https://docs.google.com/forms/d/e/1FAIpQLSdL1gArrUDUaLq4RhI_oYCMh9TO0KpoqmdJekwWr-zrWoGV7g/viewform



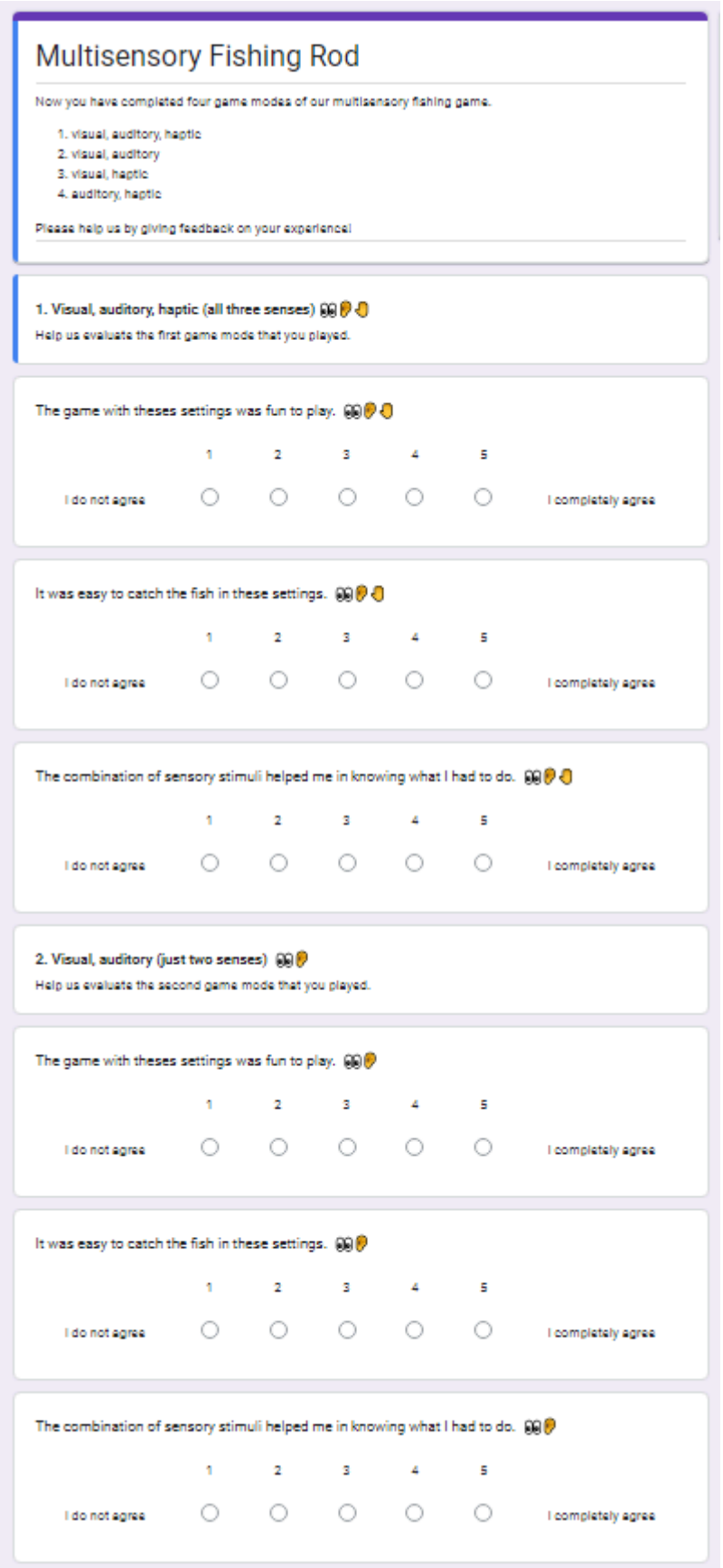
Form to schedule the experimentations with users

9.13. Information Sheet & Data collection consensus

We also inform that if you are interested in viewing the (English) form that informs the participant about the experiment, detailing its nature, the data to be collected, and providing the QR reference for the post-experiment qualitative evaluation form, along with a data usage consent form that the user is required to sign. The document, titled "**Information sheet.pdf**," can be found in our GitHub directory at the following address:

https://github.com/RiccardoCalcagno/MultisensoryFishingRod/blob/main/Information%20sheet.pdf

9.14 Questionnaire



Multisensory Fishing Rod

Now you have completed four game modes of our multisensory fishing game.

1. visual, auditory, haptic
2. visual, auditory
3. visual, haptic
4. auditory, haptic

Please help us by giving feedback on your experience!

1. Visual, auditory, haptic (all three senses)

Help us evaluate the first game mode that you played.

The game with theses settings was fun to play.

	1	2	3	4	5	
I do not agree	○	○	○	○	○	I completely agree

It was easy to catch the fish in these settings.

	1	2	3	4	5	
I do not agree	○	○	○	○	○	I completely agree

The combination of sensory stimuli helped me in knowing what I had to do.

	1	2	3	4	5	
I do not agree	○	○	○	○	○	I completely agree

2. Visual, auditory (just two senses)

Help us evaluate the second game mode that you played.

The game with theses settings was fun to play.

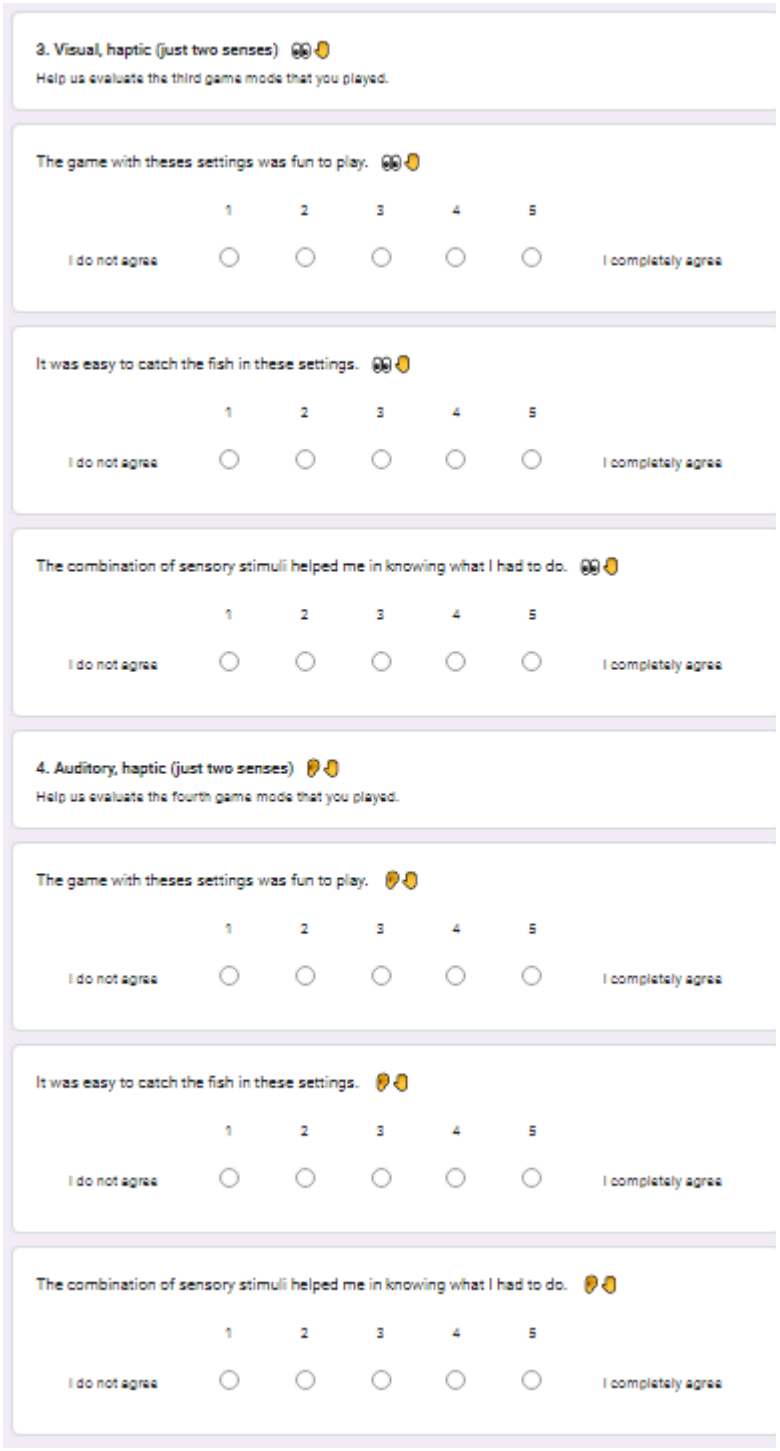
	1	2	3	4	5	
I do not agree	○	○	○	○	○	I completely agree

It was easy to catch the fish in these settings.

	1	2	3	4	5	
I do not agree	○	○	○	○	○	I completely agree

The combination of sensory stimuli helped me in knowing what I had to do.

	1	2	3	4	5	
I do not agree	○	○	○	○	○	I completely agree



9.14 Questionnaire results

User	All 3			V-A			V-H			A-H		
	fun	ease	help	fun	ease	help	fun	ease	help	fun	ease	help
1	5	4	5	4	4	5	4	4	5	5	2	5
2	5	5	5	5	4	4	5	5	5	5	5	5
3	5	4	5	5	4	5	5	4	4	5	3	4

4	4	4	5	3	4	4	3	2	3	2	1	2
5	4	4	5	4	3	5	5	5	5	4	2	4
6	5	5	5	5	5	5	4	4	4	3	2	5
7	5	3	5	5	3	5	3	2	5	1	1	5
8	5	5	5	5	5	5	5	4	3	5	5	5
9	4	2	5	3	3	3	4	3	3	2	3	2
10	5	5	5	4	5	5	4	4	5	3	3	2

9.15 Objective Game Measurements

9.15.1 Game Duration (seconds)

	All 3		V-A		V-H		A-H
Pietro	183.79373	Pietro	145.39961	Pietro	111.181114	Pietro	115.453316
Matteo	226.64943	Matteo	262.16867	Matteo	147.70792	Matteo	138.76865
Stonfer	169.2598	Davide	167.80367	Davide	85.738106	Davide	113.506805
Stonfer	143.21147	Stonfer	94.361496	Stonfer	83.708084	Chiara	116.84165
Dades	94.91193	Dades	84.901024	Dades	72.55748	Stonfer	253.13556
Cate	167.71062	Cate	164.27235	Cate	236.80803	Dades	114.63264
Taba	60.89587	Lorenzo	108.40631	Maggiano	130.22925	Cate	172.53781
Taba	135.14143	Fabio	123.98146	Lorenzo	176.26018	Maggio	187.3214
Anna	60.57245	Pietro	48.916344	Fabio	128.65234	Anna	207.72272
Anna	122.611916	Erik	79.81262	Pietro	117.867966	Lucri	132.99796
Lucri	219.76418	Simone	147.75935	Erik	132.55672	Chiara	169.34756
Anna	75.154495	Federico	106.365166	Simone	375.83362	Lorenzo	249.17912
	240.74577			Federico	181.63516	Fabio	229.00378
Lorenzo	219.59221					Pietro	157.76083
Fabio Training	185.87875					Erik	210.45895
Pietro	160.37018					Simone	192.60864
Erik	232.0642					Federico	145.26721
Simone	285.13083						
Federico	202.61758						
Mean	167.6882548		127.8456725		152.3643054		170.9732118
min	60.57245		48.916344		72.55748		113.506805
	Best for all 3:		Best between combinations:				
	60.57245		A-H	48.916344			

9.15.2 Attracting Fish Score

PlayerName	All 3		V-A		V-H		A-H
Pietro	2.335263	Pietro	-0.28375334	Pietro	2.047737	Pietro	1.1433504
Matteo	0.67463696	Matteo	4.4351387	Matteo	2.9865966	Matteo	1.0605297
Stonfer	2.2662995	Davide	2.8223643	Davide	1.0726371	Davide	1.2043965
Stonfer	3.4342632	Stonfer	0.35733697	Stonfer	2.303137	Chiara	-1.0354698
Dades	0.8000633	Dades	0.96006316	Dades	1.7588371	Stonfer	-2.838848
Cate	2.1872966	Cate	1.359637	Cate	2.31703	Dades	-3.7542632
Taba	0.21259964	Lorenzo	2.1279633	Maggiano	0.86333036	Cate	-1.1135032
Taba	0.944237	Fabio	-0.16950536	Lorenzo	3.3601966	Maggio	-0.8797698
Anna	2.4830995	Pietro	0.624881	Fabio	2.3656964	Anna	1.8804674
Anna	-0.20997	Erik	-0.42116332	Pietro	0.5828967	Lucri	-21.72647
Lucri	0.28802967	Simone	-4.2969627	Erik	2.8248963	Chiara	2.57133
Anna	1.7495632	Federico	0.9121632	Simone	7.863758	Lorenzo	0.9957447
	-3.016903			Federico	-0.5157609	Fabio	-9.423452
Lorenzo	-0.020336866					Pietro	0.2802596
Fabio Training	0.4593966					Erik	0.63461876
Pietro	1.9274955					Simone	0.38899636
Erik	2.9444008					Federico	-1.0026014
Simone	0.059443474						
Federico	-0.20909071						
Mean	1.016304598		0.7023469092		2.294691405		-1.859687293
max	3.4342632		4.4351387		7.863758		2.57133
	Best for all 3:		Best between combinations:				
	3.4342632		A-H	7.863758			

9.15.3 Hooking Fish Score

PlayerName	All 3		V-A		V-H		A-H
Pietro	-9.485714	Pietro	-5.207143	Pietro	-4.6857142	Pietro	-0.54285717
Matteo	-49.335712	Matteo	-4.4785714	Matteo	1.05	Matteo	0.2
Stonfer	-64.48572	Davide	1.2714286	Davide	0.49285713	Davide	-0.64285713
Stonfer	-2.4214287	Stonfer	1.8642857	Stonfer	3.2714286	Chiara	1.3214285
Dades	2.45	Dades	1.3071429	Dades	2.1142857	Stonfer	-4.4
Cate	-2.6214285	Cate	1.9142857	Cate	-1.807143	Dades	0.65
Taba	-2.85	Lorenzo	-6.1928573	Maggiano	-6.457143	Cate	-1.2571428
Taba	-21.37143	Fabio	-0.6857143	Lorenzo	-11.271428	Maggio	-1.1714286
Anna	1.1928571	Pietro	-1.8142858	Fabio	-4.5214286	Anna	-4.5142856
Anna	-1.4	Erik	1.9785714	Pietro	-0.2	Lucri	-0.5
Lucri	-36.721428	Simone	-2.057143	Erik	-1.3357143	Chiara	-0.70714283
Anna	-4.5	Federico	-2.1	Simone	-11.407143	Lorenzo	-11.528571
	-42.442856			Federico	0.79285717	Fabio	-4.35
Lorenzo	-43.74286					Pietro	-3.9571428
Fabio Training	-8.928572					Erik	-2.3714285
Pietro	-9.057143					Simone	-7.3714285
Erik	-5.307143					Federico	-5.5
Simone	-29.95						
Federico	-44.5						
Mean	-19.76203043		-1.183333375		-2.612637346		-2.743697437
max	2.45		1.9785714		3.2714286		1.3214285
	Best for all 3:		Best between combinations:				
	2.45		A-H	3.2714286			

9.15.4 Retrieving Fish Score

PlayerName	All 3		V-A		V-H		A-H
Pietro	1.0625983	Pietro	-0.063966304	Pietro	2.4391954	Pietro	0.17263459
Matteo	0.57177347	Matteo	-1.7625322	Matteo	1.0260522	Matteo	-0.9000893
Stonfer	3.2106671	Davide	-3.8423483	Davide	1.7991995	Davide	-0.8281866
Stonfer	1.9612331	Stonfer	-0.021826357	Stonfer	2.5975177	Chiara	-1.83387
Dades	0.5715921	Dades	1.2105068	Dades	2.9464502	Stonfer	-7.369415
Cate	-1.1983825	Cate	1.3802414	Cate	-1.2961243	Dades	0.23680124
Taba	2.2190275	Lorenzo	1.156882	Maggiano	-2.9444911	Cate	0.31091487
Taba	2.261465	Fabio	2.414765	Lorenzo	1.8657914	Maggio	-3.986542
Anna	3.7110837	Pietro	2.3862925	Fabio	2.262649	Anna	-1.2993276
Anna	-1.5585138	Erik	-0.038122356	Pietro	-0.66721	Lucri	-0.2667554
Lucri	-2.3345764	Simone	-2.0002844	Erik	0.90992963	Chiara	-0.3624804
Anna	1.8654106	Federico	0.9264276	Simone	2.7515574	Lorenzo	-3.5333333
	1.8637307			Federico	-3.8285823	Fabio	-4.6527
Lorenzo	-1.6888					Pietro	-1.7150869
Fabio Training	-0.04236263					Erik	0.33561906
Pietro	-1.9440411					Simone	-0.017928421
Erik	-2.62816					Federico	-2.1169677
Simone	-1.5170165						
Federico	1.1644146						
Mean	0.3974285916		0.1455029486		0.7586103638		-1.636865462
max	3.7110837		2.414765		2.9464502		0.33561906
	Best for all 3:		Best between combinations:				
	3.7110837		A-H	2.9464502			

9.15.5 Damaged Wire

PlayerName	All 3		V-A		V-H		A-H
Pietro	0.23278733	Pietro	1.0021678	Pietro	0.76333773	Pietro	0.8751089
Matteo	0.7329691	Matteo	1.0070692	Matteo	0.3277268	Matteo	0.2801922
Stonfer	1.0009602	Davide	1.0034726	Davide	0.5137292	Davide	1.0048825
Stonfer	0.30773613	Stonfer	1.0071446	Stonfer	0.54936177	Chiara	1.0011593
Dades	0.62136084	Dades	1.0013051	Dades	0.37166762	Stonfer	1.0059379
Cate	1.0045059	Cate	0.77867836	Cate	0.55962855	Dades	0.6885095
Taba	0.21490029	Lorenzo	0.765135	Maggiano	1.0001004	Cate	0.06442383
Taba	0.123235166	Fabio	0.03210739	Lorenzo	0.44989026	Maggio	0.85093033
Anna	0.6400596	Pietro	0.061608505	Fabio	0.13667816	Anna	0.45454764
Anna	0.6143887	Erik	0.3223442	Pietro	0.06517326	Lucri	0.5601921
Lucri	0.51468486	Simone	0.81661284	Erik	0.29055205	Chiara	0.6229572
Anna	0.5147134	Federico	0.25491616	Simone	0.44063556	Lorenzo	0
	0.5023414			Federico	0.1897377	Fabio	0.6258324

Lorenzo	0.7198069					Pietro	0.08858757
Fabio Training	0.3475029					Erik	0.603063
Pietro	0.27912834					Simone	0.21472542
Erik	0.6688257					Federico	0.26864982
Simone	0.5247557						
Federico	0.15686424						
Mean	0.5116592998		0.6710468129		0.43524762		0.5417470359
min	0.123235166		0.03210739		0.06517326		0
	Best for all 3:		Best between combinations:				
	0.123235166		A-H	0			